Titre: Title:	Modélisation mathématique et informationnelle des problèmes de tournées de véhicules dans le marquage des réseaux routiers
Auteur: Author:	Ciro Alberto Amaya Guio
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Amaya Guio, C. A. (2006). Modélisation mathématique et informationnelle des problèmes de tournées de véhicules dans le marquage des réseaux routiers [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7778/

UNIVERSITÉ DE MONTRÉAL

MODÉLISATION MATHÉMATIQUE ET INFORMATIONNELLE DES PROBLÈMES DE TOURNÉES DE VÉHICULES DANS LE MARQUAGE DES RÉSEAUX ROUTIERS

CIRO ALBERTO AMAYA GUIO

DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION

DU DIPLÔME DE PHILOSOPHIAE DOCTOR (Ph.D.)

(GÉNIE INDUSTRIEL)

DÉCEMBRE 2006

Your file Votre référence
ISBN: 978-0-494-24535-4
Our file Notre référence
ISBN: 978-0-494-24535-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MODÉLISATION MATHÉMATIQUE ET INFORMATIONNELLE

DES PROBLÈMES DE TOURNÉES DE VÉHICULES

DANS LE MARQUAGE DES RÉSEAUX ROUTIERS

présentée par: AMAYA GUIO Ciro Alberto

en vue de l'obtention du diplôme de: Philosophiae Doctor

a été dûment acceptée par le jury d'examen constitué de :

M. ROUSSEAU Louis-Martin, Ph.D., président

M. TRÉPANIER Martin, ing., Ph.D., membre et directeur de recherche

M. LANGEVIN André, Ph.D., membre et codirecteur de recherche

M. GAMACHE Michel, ing., Ph.D., membre

M. RENAUD Jacques, Ph.D., membre externe

À mon père et mère qui m'ont vu commencer mais qui ne sont pas restés pour me voir finir.

À Sandra, Alejandro, Carolina et Nathalia.

REMERCIEMENTS

Je tiens à remercier mes directeurs, les professeurs Martin Trépanier et André Langevin. Mes sincères remerciements à Sandra, ma très chère épouse, pour son appui inconditionnel. Mes remerciements à tous les membres du groupe Polygistique et tous ceux qui de près ou de loin ont permis à ce projet de voir le jour. Un merci spécial à mon fils Alejandro qui se réveillait tôt pour « travailler » avec moi.

RÉSUMÉ

Dans cette thèse, nous étudions trois problèmes de tournées de véhicules associés aux opérations de marquage des réseaux routiers. Les problèmes sont étudiés selon les points de vue mathématique et informationnel. Plus précisément, dans la partie mathématique nous étudions les problèmes à l'aide d'heuristiques et de la programmation mathématique en nombres entiers. Dans la partie informationnelle, nous modélisons le problème à l'aide de l'approche orientée-objet en transport.

Nous présentons tout d'abord les activités de marquage routier de façon générale. Nous y distinguons trois niveaux de gestion complémentaires : l'étape de la planification générale du réseau routier, l'étape de la planification saisonnière et finalement, au niveau opérationnel, l'étape du suivi des activités réalisées. Nous traitons plus spécifiquement du problème de génération de tournées pour la planification saisonnière, mais les outils développés durant cette thèse peuvent également servir à l'étape de suivi.

La première variante du problème de marquage est une adaptation des problèmes mixtes de localisation et de tournées sur les arcs (PLTA), que nous appelons le "problème de tournées sur les arcs avec points d'arrêt pour remplissage". Dans ce problème, nous avons un véhicule marqueur (VM) de capacité limitée qui doit desservir des arcs (tracer des lignes de signalisation sur la chaussée). Le réservoir du VM est rempli sur place par un autre véhicule (le véhicule de réapprovisionnement, VR) qui doit retourner au dépôt après chaque remplissage. L'objectif poursuivi consiste à minimiser le coût total des tournées des deux véhicules. Un modèle de programmation linéaire en nombres entiers est développé. Le modèle mathématique est construit sur un graphe mixte inspiré des modèles du CARP (*capacitated arc routing problem*). L'ensemble des arêtes est transformé en arcs en remplaçant chaque arête par deux arcs opposés, et en ajoutant des contraintes pour restreindre le passage de service dans une seule des deux directions. Pour résoudre le problème, le graphe a été augmenté en ajoutant deux ensembles de liens

qui relient chaque nœud au dépôt dans les deux directions. Des instances du problème ont été résolues à l'aide d'un algorithme de coupes. Cet algorithme fait une relaxation des contraintes de connectivité. À chaque itération, les contraintes qui ont été violées sont ajoutées jusqu'à l'obtention d'une solution réalisable. La méthode proposée est capable de résoudre en temps acceptable des problèmes de petite taille pour l'ensemble des problèmes testés sur des graphes mixtes et des problèmes de petite et moyenne taille pour l'ensemble des problèmes sur des graphes orientés.

Le deuxième problème étudié est une variante du premier problème. Dans ce cas, le véhicule de réapprovisionnement (VR) n'a pas de restriction sur le nombre de fois qu'il peut rencontrer le véhicule de marquage (VM) avant de retourner au dépôt. Un modèle de programmation linéaire en nombres entiers et une méthode de résolution heuristique sont présentés. La procédure heuristique développée consiste en trois étapes : trouver une tournée « géante » pour le VM, diviser la tournée en sections, où chacune est réalisable pour le VM, et sélectionner l'ensemble de sections satisfaisant l'autonomie du VM à un coût minimal. Des variantes de la méthode heuristique de base sont également présentées. Celles-ci permettent de trouver de solutions plus rapidement sans compromettre de plus de 4% le coût total dans les problèmes testés.

La troisième situation étudiée est une généralisation des deux autres problèmes. Dans ce cas, nous disposons de plusieurs VM et plusieurs VR. Un modèle de programmation linéaire en nombres entiers est présenté et une heuristique est développée. Cette heuristique consiste en trois étapes : dans la première étape, un problème de partitionnement est considéré; dans la deuxième étape, nous déterminons des trajets et des points de remplissage pour les VM ; et dans la troisième étape nous concevons les tournées pour le VR.

Dans la dernière partie de la thèse, nous présentons la librairie orientée-objets utilisée pour étudier les problèmes. Cette librairie encapsule les algorithmes de résolution des multiples problèmes de tournées de véhicules tels que le problème de tournées sur les arcs avec points d'arrêt pour remplissage, ce qui implique également la présence de tous

les algorithmes sous-jacents (plus court chemin, tournées sur arcs, etc.). Cette librairie est en cours d'implantation au Ministère des Transports du Québec (MTQ).

En perspective, le type de problèmes étudiés dans cette thèse ouvre la voie à plusieurs problèmes intéressants dans le domaine de l'entretien des réseaux routiers et d'autres domaines comportant des tournées sur arcs à plusieurs véhicules. Une valeur ajoutée intéressante est apportée par l'intégration d'algorithmes de recherche opérationnelle dans le système d'information afin que ces algorithmes soient pleinement utilisables par les partenaires industriels.

ABSTRACT

This dissertation addresses three problems related to the road network marking. These problems are studied from a mathematical and a computational point of view. In the mathematical part, linear integer programming formulations and heuristics are presented. The oriented object approach for transport is used for modeling the problems.

First of all, this work presents the road marking activities in a general context. Three complementary management levels are distinguished: the stage of general planning for the road network, the stage of seasonal planning, and finally, at the operational level, the stage of the completed activities follow-up. Specifically, this project addresses the vehicle routing problem for the seasonal planning stage, but the tools developed in this thesis can also be used at the follow-up stage.

The first problem studied is an extension of location-arc routing problems, which from here on will be called "capacitated arc routing problem with refill points". In this problem, we have a limited capacity vehicle which must serve arcs (to trace lines on the roadway). The servicing vehicle (SV) is filled on the spot by another vehicle (the refilling vehicle, RV) which must return to the depot after each filling. The aim is to minimize the total routing cost of the two vehicles. A linear integer model is developed. The mathematical model is built on a mixed graph, based on the models for the CARP (capacitated arc routing problem). The edges are transformed into arcs by replacing each edge by two opposed arcs, and by adding constraints to restrict the passage of service in only one of the two directions. To solve the problem, the graph was augmented by adding two sets of arcs connecting each node with the depot and the depot with each node. Problem instances were solved using a cutting plane algorithm. This algorithm makes a relaxation of connectivity. At each iteration, the violated are added until obtaining a feasible solution. The suggested method is able to solve, in acceptable time,

small size problems on mixed graphs and small or medium sized problems on directed graphs.

The second problem studied is an extension of the first one. In this case, the refilling vehicle does not have a restriction on the number of times that it can meet the servicing vehicle (SV) before returning to the depot. A linear integer model and a heuristic are presented. The developed heuristic procedure consists of three stages: find a giant tour for SV, divide the tour into sections, where each one is feasible for SV, and select the sections satisfying the autonomy of SV at a minimal cost. Variations to the basic heuristic method are also presented. Given these variations, it is possible to quickly find solutions compromising no more than 4% of the total cost in the tested problems.

The third situation studied is a generalization of the other two problems. This case considers multiple servicing and refilling vehicles. A linear integer model is presented and a heuristic method is proposed. This heuristic consists of three stages: in the first stage, a generalized assignment problem is considered; in the second stage, the refill points and trajectories are determined for the servicing vehicles; and in the third stage the routings for the refilling vehicles are considered.

The last section presents the object oriented library developed to study the problems. This library includes solutions to several vehicles routing problems such as the capacitated arc routing problem with refill points, which also implies the presence of all the subjacent algorithms (shortest path, arc routing, etc).

Finally, one cannot doubt the importance of the problems studied in this thesis as an opening way to several interesting problems in the field of road marking maintenance and in other fields, and as an adding-value tool for the integration of operations research algorithms within information systems in order to optimize the resources allocated to the road network maintenance.

TABLE DES MATIÈRES

LISTE DES *TABLES* ET TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

CARP : *Capacitated Arc Routing Problem*

CARP-RP : *Capacitated Arc Routing Problem with Refill Points*

FHA : *Federal Highway Administration*

GIS-T : *Geographic Information System for Transportation*

LARP : *Location-arc routing problems*

MOOT : Modélisation orientée objet en transport

MTQ : Ministère des Transports du Québec

PFC : Problème du facteur chinois

PFR : Problème du facteur rural

PLTA : Problèmes mixtes de localisation et de tournées sur les arcs

PRE : Problème de recouvrement d'ensembles

PTA : Problèmes de tournées sur les arcs

PTG : Problème de tournée généralisé

PTS : Problèmes de tournées sur les sommets

RV : *Refilling vehicle*

RO : Recherche opérationnelle

SAD : Systèmes de planification et d'aide à la prise de décisions

SIG : Systèmes d'information géographique

SIGR : Systèmes d'information météorologique routière

SIG-T : Systèmes d'information géographiques pour le transport

SV : *Servicing Vehicle*

UML : *Unified Modeling Language*

VM : Véhicule de marquage

VR : Véhicule de réapprovisionnement

LISTE DES ANNEXES

INTRODUCTION

Les problèmes liés au transport n'apparaissent pas seulement dans le transport de personnes ou de marchandises mais aussi dans des situations de prestation de services qui doivent être assurés auprès de clients. On retrouve une telle situation dans les travaux d'entretien des infrastructures de transport.

La signalisation routière est une activité d'entretien routier qui requiert de nombreuses ressources et une bonne gestion permet de maintenir le réseau dans un état donné avec des économies appréciables.

Une des activités de la planification du marquage routier est la construction de circuits pour tracer les lignes sur la chaussée. Au Québec, par exemple, le Ministère des Transports du Québec (MTQ), est le responsable de la signalisation et dispose de différentes ressources et d'un inventaire de routes du réseau de transport routier à être peintes. Les défis dans la planification sont multiples, puisqu'on doit tenir compte de problèmes d'ordre technologique, informatique et mathématique.

Une planification judicieuse apporte des réductions des sommes requises pour maintenir le réseau dans un état donné, et une gestion efficiente mène à l'amélioration significative du marquage routier.

Le marquage routier

Si on regarde les ressources financières utilisées pour l'entretien des équipements de sécurité au Québec en 2004-2005 (MTQ, 2006) on peut remarquer, comme le rapporte le MTQ dans le tableau suivant, que le marquage de la chaussée est le coût le plus important.

Tableau 0.1. Entretien des équipements de sécurité au Québec en 2004-2005

Source	Coût
Le marquage de la chaussée	18,8 M$
La signalisation routière	10,9 M$
Les dispositifs de retenue	7,5 M$
L'éclairage	3,9 M$
Les feux lumineux	2,3 M$
Total	**43,4 M$**

Les activités de marquages dépendent du type de route à marquer. Au Québec, le Ministère des Transports (MTQ, 2002a) classifie les routes en :

- autoroutes ;
- routes nationales ;
- routes régionales ;
- routes collectrices ;
- chemins ;
- routes locales de niveau 1 ;
- routes locales de niveau 2 ;
- routes locales de niveau 3.

De manière générale, la planification des travaux pour le marquage sur un réseau donné consiste à déterminer comment et quelles voies (ou secteurs de voies) doivent être marquées (peintes) sur chaque élément du réseau, ainsi que le moment pour le faire. Les travaux consistent à appliquer sur les chaussées des marques, conformes aux dessins normalisés (FHA, 2001 ; MTQ, 2002b), qui facilitent le guidage de l'automobiliste, améliorent le flux de la circulation et contribuent au confort et à la sécurité routière. On distingue trois niveaux ou étapes de gestion complémentaires dans la gestion du marquage (Figure 0.1) :

- l'étape de la planification générale du réseau routier. Dans cette étape, on décide les sens ou côtés des routes, et les routes qui seront marquées à chaque saison à venir. D'une manière générale, on divise le réseau en routes (segments de routes) qui doivent être marquées chaque année, en routes (sens de routes, segments de routes) qui doivent être peintes toutes les deux années, et routes qui doivent être marquées à intervalle plus long ;
- l'étape de la planification saisonnière : à cette étape, on choisit les interventions exactes à exécuter au cours d'une période sur un réseau régional dans le contexte d'un horizon de planification d'environ 6 mois (de avril à septembre) ;
- et finalement au niveau opérationnel, on a une étape de contrôle du suivi. Dans cette étape, on doit modifier des circuits de façon dynamique afin de répondre à des changements survenus la dernière minute et ajuster les différences entre le travail prévu et celui réalisé.

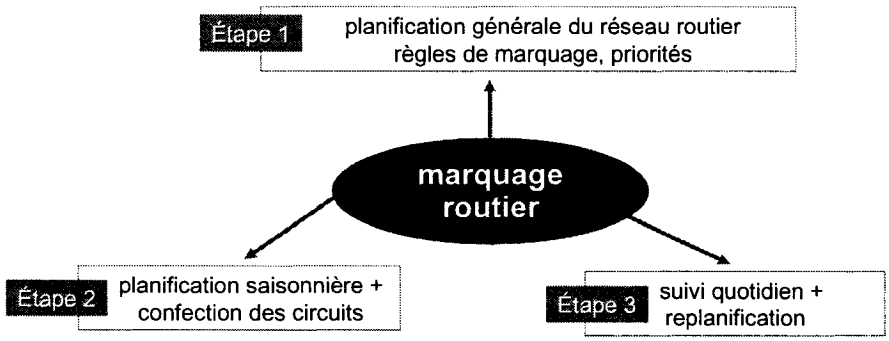


Figure 0.1. Étapes de la planification du marquage.

La définition du temps de réfection du marquage peut se faire à l'aide de modèles mathématiques. Par exemple, Kouskoulas (1988) et Easa (1991) ont proposé un modèle analytique pour l'optimisation de la fréquence du marquage sur un réseau donné. Le modèle utilise des paramètres tels que l'apparence des marques, la réfection de chaussée, la durabilité, la luminosité, la longueur des routes, le volume du trafic, la vitesse du trafic, etc.

Au niveau opérationnel, il existe plusieurs facteurs qui font de la planification du marquage routier un travail complexe. Par exemple, on peut citer les facteurs tels que le type de peinture, le nombre et le type de lignes à tracer, la couleur à utiliser, la longueur et le sens de traçage, les types de voies, les priorités de marquage, le taux d'application de peinture, l'état de durabilité des voies, les conditions météorologiques et routières, les horaires de travail et les périodes de vacances du personnel, etc. Le Canada utilise deux types de couleurs à la différence d'autres pays qui utilisent seulement la couleur blanche (voir Hawkins, et al. (2002)). La prise d'inventaire est le squelette du système à partir duquel les planificateurs peuvent construire les circuits et déterminer les besoins.

Les modèles de planification requièrent de nombreux paramètres sur l'état du réseau et sur l'évolution de ses éléments en fonction des interventions effectuées. Si les valeurs de ces paramètres sont inexactes, on risque d'obtenir des plans d'entretien inappropriés ou inutilement coûteux. Des développements dans le domaine du traitement de l'information, tels que les systèmes d'information géographique (SIG), systèmes d'information météorologique routier (SIMR) et des systèmes de positionnement global (SPG) permettent non seulement leur utilisation dans une gestion en temps réel, mais également une utilisation plus poussée des algorithmes de planification. L'intégration de ces technologies présente certaines difficultés, notamment dans le traitement et l'intégration de nombreuses et différentes sources de données. Une attention toute particulière doit être portée aux spécificités et aux contraintes du réseau et du transport routier dans la modélisation et le traitement de l'information.

Méthodologie

Dans un premier temps, une définition globale du problème est élaborée au niveau informationnel à l'aide d'un modèle objet servant à définir, mettre en forme et offrir une méthode de résolution du problème de planification des activités. Ensuite nous nous concentrerons sur deux problèmes liés aux activités de marquage. Pour ces deux problèmes des modèles mathématiques de programmation en nombres entiers sont

proposés. Dans les deux cas, un certain nombre de contraintes doivent être satisfaites parmi lesquelles des contraintes d'autonomie des véhicules traceurs de lignes. Nous présentons une méthodologie exacte de résolution qui sera utile pour des problèmes de petite taille, mais pour des réseaux de taille importante, la méthode est peu efficace. Donc, nous développons une méthode heuristique.

Par la suite, nous précisons les éléments utilisés lors de l'intégration. L'intégration des données opérationnelles et des algorithmes mathématiques a été conçue à partir d'une librairie informationnelle. Dans le chapitre 6, l'architecture et la fonctionnalité générale de la bibliothèque sont décrites.

Organisation du document

Ce document est organisé comme suit. Après cette introduction, le premier chapitre présente une synthèse des contributions de cette thèse. Ensuite, au chapitre 2, la revue de littérature est divisée de deux parties. La première partie traite de l'approche mathématique de recherche opérationnelle pour résoudre des problèmes mixtes de localisation et de tournées sur les arcs (PLTA ou en anglais : *location-arc routing problems*, LARP). Ce type de problèmes se retrouve dans des contextes où on doit déterminer simultanément un parcours sur un sous-ensemble d'arcs dans un réseau, ainsi que la localisation de certaines installations. La deuxième partie présente des outils pour la modélisation informationnelle telle que l'approche orientée-objet en transport.

Le chapitre 3 étudie le problème de tournées sur les arcs avec capacité et points de remplissage (PTAC-PR). Cette situation est caractéristique des opérations de marquage où le planificateur dispose d'un véhicule pour faire le service, ce véhicule a une capacité finie et doit être rempli sur place par un autre véhicule. Le problème consiste à déterminer simultanément les tournées à suivre par les deux véhicules de façon à ce que le coût total soit minimisé. Un programme linéaire en nombres entiers est proposé et des tests sont réalisés sur plusieurs réseaux.

Dans le chapitre 4, une amélioration du modèle exposé dans le chapitre précédent est présentée. Le chapitre présente un modèle de programmation mathématique et une méthode de résolution heuristique, cette méthode permettant de résoudre des problèmes réalistes.

Le chapitre 5 présente le problème de tournées sur les arcs avec points de remplissage avec plusieurs véhicules citerne (véhicules de réapprovisionnement) et plusieurs véhicules de marquage.

Au chapitre 6, on présente l'outil informationnel développé qui permet l'intégration des données opérationnelles et adapte des outils mathématiques existants afin de fournir des solutions opérationnelles aux usagers. Les conclusions de la recherche viennent clore le document.

CHAPITRE 1 : SYNTHÈSE DES CONTRIBUTIONS

La grande contribution de cette thèses est la présentation, la modélisation et la résolution d'un problème qui, à notre connaissance, n'a pas été étudié scientifiquement : le problème des tournées de véhicules avec remplissage pour le marquage routier. Nous avons étudié le problème de façon mathématique en regardant trois situations particulières qui sont présentées dans les chapitres 3, 4 et 5. Le chapitre 6, pour sa part, présente l'outil informatique développé, ce qui permet d'approcher le problème de façon informationnelle.

La contribution du troisième chapitre est de présenter un modèle pour le problème de tournées sur les arcs avec points d'arrêt pour remplissage. Dans ce chapitre, un modèle de programmation linéaire en nombres entiers et une méthode de résolution exacte sont donnés. Les résultats montrent que la méthode de résolution permet de résoudre de problèmes de petite et moyenne taille. Cette contribution prend la forme d'un article accepté pour publication dans la revue *Operations Research Letters*.

La contribution du quatrième chapitre est de présenter un modèle qui incorpore une nouvelle facette du problème de tournées sur les arcs avec points d'arrêt pour remplissage et de développer la première méthode heuristique pour résoudre ce problème en particulier. Les résultats obtenus montrent que l'heuristique de base peut résoudre des problèmes de taille importante et que des variations à la méthode permettent de résoudre plus rapidement le problème sans compromettre de façon importante le coût total. Cette contribution prend la forme d'un article soumis à la revue *Journal of the Operational Research Society*.

La contribution du cinquième chapitre est de présenter un modèle pour le problème de tournées sur les arcs avec points d'arrêt pour remplissage depuis plusieurs véhicules.

Dans ce chapitre, un modèle de programmation linéaire en nombres entiers et une méthode de résolution heuristique sont donnés.

La dernière contribution, présentée au chapitre 6, est le développement d'un outil informationnel orienté-objet qui combine des éléments mathématiques provenant de la recherche opérationnelle et des éléments d'accès à des bases de données pour l'intégration des données opérationnelles. Cet outil est une librairie informatique qui encapsule la solution de multiples problèmes de tournées de véhicules tels que le problème de tournées sur les arcs avec points d'arrêt pour remplissage. La librairie informatique est constituée de plus de 40 classes qui permettent la manipulation de réseaux routiers en les modélisant comme *graphes*. Une fois compilées, les fonctions de la librairie sont accessibles depuis la grande majorité des logiciels implantés sur Windows, ainsi que depuis la plupart des environnements de développement. La librairie a été implantée dans le système informationnel du MTQ.

CHAPITRE 2 : REVUE DE LITTÉRATURE

Dans cette thèse, nous nous intéressons à la modélisation du problème de marquage routier (PMR). La planification de la signalisation routière est composée de plusieurs activités, mais pour l'objet de notre recherche, nous sommes intéressés par le marquage de la chaussée (peinture). Pour représenter le problème, nous avons établi trois étapes qui décrivent la situation (voir chapitre précédent). Dans chacune de ces étapes, on doit prendre des décisions qui peuvent affecter l'efficacité des opérations du marquage. Étant donné que le problème en général est très complexe, cette thèse se concentre sur la deuxième étape, c'est-à-dire l'étape de génération des tournées pour le marquage.

Le problème de marquage routier, à notre connaissance, n'a pas été étudié par d'autres chercheurs, ou au moins il n'a pas été publié dans des revues scientifiques. On trouve des publications pour des problèmes reliés au marquage de réseaux, tel que la définition du temps de réfection du marquage (par exemple Kouskoulas (1988) ou Easa (1991)) mais aucune travaille sur le problème de génération de tournées. Cependant, le problème de marquage routier est une généralisation du problème de tournées sur les arcs et il peut être vu comme un problème de localisation (LARP) (voir chapitres 3, 4 et 5). Pour cette raison, nous nous sommes guidés sur les méthodes provenant de la recherche opérationnelle pour la modélisation mathématique. Ce volet a été exploré par plusieurs auteurs qui ont modélisé le problème de la planification d'interventions comme un problème de tournées de véhicules sur les arcs. Ainsi, la première partie de notre revue porte essentiellement sur la construction des tournées sur les arcs et sur la composante mixte de localisation et tournées. Une attention particulière est faite au problème de tournées sur les arcs (*CARP* ou *Capacitated Arc Routing Problem*).

Pour la modélisation informationnelle, on doit tenir compte qu'une solution est efficace dans la mesure où l'on obtient une intégration complète entre les méthodes mathématiques et celles de l'informatique. La modélisation orientée-objet se veut un

outil d'analyse et de résolution de ce problème. Cette approche permet d'intégrer les différents aspects technologiques, le système informationnel et les algorithmes mathématiques. Ainsi, la deuxième partie de cette revue porte sur l'approche orientée-objet en transport.

2.1 Problèmes de tournées de véhicules et problèmes de localisation

Dans le domaine des transports, il existe des problèmes qui consistent non seulement à transporter des personnes ou des marchandises, mais aussi à parcourir les routes comme dans les cas d'entretien des infrastructures routières. Pour simplifier, nous utiliserons le mot « client » pour indiquer les endroits à desservir. L'élaboration de tournées de véhicules consiste à déterminer un itinéraire à coût minimal, que doivent suivre des véhicules dans le but de donner un service.

Quant à la façon de modéliser les clients à desservir, la littérature parle principalement de deux grandes familles de problèmes : les problèmes de tournées sur les sommets (PTS), dans lesquels les clients sont associés aux sommets d'un réseau; et les problèmes de tournées sur les arcs (PTA), dans lesquels les clients sont associés aux segments d'un réseau (appelés arcs si orientés et arêtes si non orientés). Une troisième famille de problèmes, regroupe les deux familles antérieures dans un seul problème plus général, qui est le problème de tournée généralisé (PTG). Dans le PTG on a un graphe G= (N, L = A$\cup$E) connexe et sans cycles, où N est l'ensemble des nœuds et L est l'ensemble des liens (segments), celui-ci formé par l'union de l'ensemble A des arcs et l'ensemble E des arêtes. Avec chaque lien, on a un coût (distance, longueur, temps) non négatif. Le but du PTG est de minimiser le coût d'un circuit traversant un sous-ensemble L' de liens obligatoires (L' = (A'$\cup$E') $\subseteq$ L) et un sous-ensemble N' de nœuds obligatoires (N'$\subseteq$ N). Le circuit peut aussi utiliser d'autres arcs et d'autres nœuds dans le graphe (voir : Orloff (1974), Blais & Laporte (2002), Corberán, et al. (2003)).

L'objectif du problème de localisation est de trouver des endroits pour placer des installations de façon à optimiser un certain objectif. Celui-ci est souvent lié à la distance

à un ensemble d'installations déjà existantes. Cet objectif est habituellement la minimisation de la somme totale de distances ou la minimisation de la distance de l'installation la plus éloignée. Eiselt & Laporte (1995) divisent les problèmes de localisation en trois catégories :

- problèmes de localisation continue. Pour ce type de problèmes, l'ensemble des solutions réalisables est un plan n-dimensionnel, $\Re^n$;
- problèmes de localisation discrets. L'ensemble d'emplacements potentiels est alors indiqué ;
- problèmes sur des réseaux. Les installations doivent se localiser sur un graphe, sur des nœuds ou sur des arcs. Les installations peuvent se modéliser soit comme points ou soit comme structures plus complexes tel que des chemins, arbres ou cycles (Labbé, et al., 1998).

Le but de cette section est de faire une revue de littérature sur le problème mixte de localisation et de tournées. Auparavant, nous présentons un survol du problème de tournées sur les arcs.

2.1.1 Problème de tournées sur les arcs

Dans un PTG, si seuls les liens sont obligatoires, on obtient la classe de problèmes dite de tournée sur les liens, c'est-à-dire les problèmes de tournée sur les arcs et arêtes (PTA). Les PTA apparaissent dans de nombreux contextes, ils ont eu beaucoup d'attention des chercheurs, voir : Eiselt, et al. (1995) et Dror (2000). Les deux problèmes classiques du PTA sont le problème du facteur chinois (PFC) et le problème du facteur rural (PFR). Dans les deux cas, on a N' = $\varnothing$ et pour le premier, on a L' = L. Le PFC peut être résolu à l'optimum en un temps polynomial dans les cas où le graphe est strictement orienté ou strictement non orienté. Cependant, le PFC mixte est NP-complet. Le PFR apparaît naturellement dans les applications concernant les réseaux routiers, par exemple dans les activités d'entretien. Il consiste à déterminer une tournée de longueur minimale passant

au moins une fois sur chaque lien requérant un service. Dans le cas seulement où le graphe défini par les liens obligatoires et leurs sommets incidents est connexe, alors le problème peut être résolu en temps polynomial. Autrement, il est NP-difficile, voir Lenstra & Rinnooy (1976) pour les cas orienté et non orienté et Frederickson, et al. (1978) pour le cas mixte.

Les graphes eulériens[1] sont la base des solutions aux problèmes PFC. Il existe des conditions nécessaires et suffisantes pour qu'un tel graphe soit eulérien. Le graphe doit être connexe dans le cas non orienté et fortement connexe dans le cas orienté. Les conditions sont connues sous le nom de conditions d'unicursalité Ford & Fulkerson (1963) :

- cas non orienté : chaque sommet doit avoir un degré pair ;
- cas orienté : le nombre d'arcs entrants doit être égal au nombre d'arcs sortants ;
- cas mixte : chaque sommet doit être incident à un nombre pair d'arcs et d'arêtes. Pour chaque sous-ensemble S de V, la différence entre le nombre d'arcs orientés de S vers (V \ S) et de (V\S) vers S ne doit pas être supérieur au nombre d'arêtes non orientées reliant S et (V \ S).

Pour parcourir un graphe eulérien, il existe plusieurs méthodes. Pour le cas non orienté la méthode la plus connue est la méthode d'*End-Pairing* d'Edmonds & Johnson (1973) :

- étape 1 : partir d'un point quelconque du graphe et déterminer un premier cycle. Si toutes les arêtes ont été visitées, terminer ;
- étape 2 : choisir une arête non visitée et incidente au cycle déjà construit. Construire un deuxième cycle comme à l'étape 1 ;

---

[1] Un graphe est eulérien s'il existe une façon de former un cycle qui passe exactement une fois sur chaque lien.

- étape 3 : fusionner les deux cycles. Retourner à l'étape 2 tant qu'il a encore des arêtes non visitées.

L'algorithme peut s'adapter pour le cas orienté. Il suffit de considérer la direction des arcs lors de la création des cycles.

Si le graphe n'est pas eulérien, il peut être parcouru à coût minimal. Le graphe doit être transformé à un graphe eulérien en résolvant un problème de l'augmentation minimale, en ajoutant des liens qui respectent les conditions d'unicursalité. Cela se réduit à résoudre un problème de transport pour le cas orienté et un de couplage pour le cas non orienté (Edmonds & Johnson, 1973).

2.1.1.1 Problème de tournées sur les arcs avec contraintes de capacité

Une extension naturelle des PTA est le problème de tournées sur les arcs avec contraintes de capacité (PTAC, ou *CARP Capacitated Arc Routing Problem*) introduit par Golden & Wong (1981). Lorsqu'on associe une demande à chaque arc à desservir, il est souvent nécessaire d'effectuer le service sur les arcs à l'aide de plusieurs véhicules, chaque véhicule étant basé à un dépôt et n'ayant qu'une capacité limitée. Le cas non orienté concerne des routes traitables en un seul passage et dans n'importe quel sens. On peut aussi considérer un PTAC orienté, où un arc désigne une route ou un côté de route avec un sens de traitement imposé. Christofides (1973) étudie une variante de ce problème. Dans ce cas, la demande sur chaque arc est strictement positive. Le problème est, alors, le problème du facteur chinois avec capacité (PFC-C).

Le PTAC généralise le PFR et plusieurs heuristiques ont été développées pour ce problème NP- dur. Hertz & Mittaz (2000) présentent et classifient ces heuristiques en heuristiques constructives d'une phase et de deux phases, algorithmes consistant à fabriquer une grande tournée en premier et grouper en second « *route-first, cluster-second* », algorithmes consistant à grouper en premier et fabriquer chaque tournée en second «*cluster-first, route-second* », et les métaheuristiques.

Parmi les heuristiques d'une phase (constructives) nous pouvons citer les suivantes :

Construct-Strike : proposée par Christofides (1973). La méthode consiste à construire progressivement des cycles qui sont enlevés du graphe si ce dernier reste connexe.

Path-Scanning : proposée par Golden, et al. (1983). Est une méthode gloutonne qui construit des tournées une par une. Chaque tournée part du dépôt et est prolongée à chaque itération vers une arête à traiter compatible avec la capacité résiduelle du véhicule. La construction d'une tournée se fait arête par arête en ajoutant chaque fois l'arête à desservir satisfaisant le mieux un critère fixé (critères de minimisation de coûts et de maximisation de la productivité). L'algorithme est exécuté une fois par critère et donne à la fin la meilleure solution.

Augment-Merge : proposée par Golden, et al. (1983). La méthode s'inspire de celle de Clarke & Wright (1964) pour les problèmes de tournées sur les nœuds. On part d'une solution où chaque tournée ne dessert qu'un seul nœud. Le nombre de véhicules utilisés est progressivement réduit grâce à deux phases : *Augment* et *Merge*. Tout d'abord, le service de certaines arêtes est transféré d'une tournée vers une autre. Ensuite, de nouvelles tournées admissibles sont ensuite obtenues en fusionnant deux tournées.

Parallel-Insert : développée par Chapleau, et al. (1984). La méthode se base sur l'idée de Path-Scanning. La méthode commence par construire une tournée avec l'arête requise la plus éloignée du dépôt. Puis deux stratégies d'insertion sont utilisées : la première détermine, étant donné une arête, la tournée qui va l'accueillir ainsi que la position d'insertion dans cette tournée, de façon à minimiser le détour provoqué par l'insertion, en respectant les contraintes de capacité et d'autonomie. La deuxième stratégie est appliquée quand tous les véhicules sont utilisés. Elle consiste à déterminer, étant donné une tournée, quel client non desservi doit y être inséré.

Augmented-Insert *:* cet algorithme est le plus représentatif de la génération d'heuristiques constructives ; il est proposé par Pearn (1991) et combine les principes utilisés dans l'algorithme *Augmented-Merge* de Golden & Wong (1981) et dans *Parallel_Insert* de Chapleau, et al. (1984). Dans une première étape, l'algorithme construit des tournées admissibles en utilisant une approche similaire à celle de Golden,

et al. (1983) laquelle est inspirée de celle développée par Clarke & Wright (1964) pour le problème de tournées sur les sommets avec capacité : partant d'un ensemble où chaque tournée ne dessert qu'une seule arête, le nombre de véhicules utilisés est progressivement réduit. Dans la deuxième étape, les arêtes qui ne sont pas traitées après cette première phase sont ensuite insérées séquentiellement dans des tournées d'une façon similaire à la stratégie d'insertion d'arêtes développée par Chapleau, et al. (1984). On doit alors déterminer la tournée existante dans laquelle il faut insérer l'arête de façon à minimiser le détour provoqué par l'insertion.

Pearn donne deux versions de l'algorithme, ci-après on transcrit la première version d'après Mittaz (2000). La version deux utilise à l'étape 2 un cycle de moindre demande au lieu d'un cycle de moindre coût.

Algorithme : AUGMENT INSERT

ENTRÉE : Un graphe non orienté G = (V, E)

SORTIE : Une solution (pas forcément optimale) du PFC dans G.

Étape 1: Soit SCij la plus courte chaîne entre les sommets v_i et v_j dans G, et soit sc_{ij} sa longueur. Pour chaque arête (v_i,v_j) de G définir Dij = $sc_{0i}+sc_{j0}$.

Poser G'=G.

Étape 2 : Déterminer l'arête (v_i,v_j) dans G' qui a la plus grande valeur de D_{ij} et qui est contenue dans un cycle de G' dans lequel se trouve également le dépôt.

Si une telle arête n'existe pas aller à l'étape 3. Sinon déterminer le cycle de moindre coût T dans G' contenant (v_i,v_j) et le dépôt, et considérer T comme une tournée ne desservant que (v_i,v_j).

Aussi longtemps que la capacité W des véhicules n'est pas dépassée et en considérant les arêtes (v_a,v_b) selon l'ordre décroissant des D_{ab}, changer le statut des arêtes (v_a,v_b) de T qui sont traversées sans être desservies.

Figure 2.1. Algorithme de Pearn

Soit v_p et v_q le premier et le dernier sommet de T incident à une arête desservie. Remplacer la chaîne reliant v_0 à v_p et la chaîne reliant v_q à v_0 par respectivement SC_{0p} et SC_{q0}. Enlever les arêtes desservies de G'. Si toutes les arêtes de G sont desservies alors stop, sinon répéter l'étape 2.

Étape 3 : Déterminer l'arête non desservie (v_i,v_j) qui a la plus grande valeur de D_{ij} et créer une tournée $T = SC_{0i} + (v_i,v_j) + SC_{j0}$ où (v_i,v_j) est la seule arête desservie.

Étape 4 : Soit E' l'ensemble des arêtes non desservies pouvant être ajoutées à T sans provoquer un dépassement de capacité et soit (v_a,v_b) l'arête de E' dont la valeur D_{ab} associée est maximum.

Calculer le détour occasionné par l'insertion de (v_a,v_b) entre le dépôt et le premier (ou le dernier) sommet incident à une arête desservie dans T. Si le coût de cette insertion est supérieur à une borne fixée B alors ôter (v_a,v_b) de E'. Sinon insérer (v_a,v_b) dans T.

Répéter l'étape 4 jusqu'à ce que E' soit vide ou que les deux arêtes incidentes au dépôt dans T soient desservies.

Étape 5. Enlever les arêtes desservies de G'.

Si G' est vide stop. Sinon aller à l'étape 3.

Figure 2.1. Algorithme de Pearn (suite)

Parmi les heuristiques à deux phases nous pouvons citer les suivantes :

Heuristique de Benavent, et al. (1990) : cette heuristique est une méthode de type *Cluster-First, Route-Second.* La première phase appelée *Cycle-Assignment*, partitionne les arêtes requises entre les véhicules en résolvant un problème d'affectation généralisée. Dans la deuxième phase, on construit une tournée pour la résolution du problème du facteur rural (PFR) pour chaque véhicule. La méthode s'inspire de l'algorithme proposé par Ficher & Jaikumar (1981) pour le VRP.

Algorithme d'Ulusoy (1985) : cette heuristique est une méthode de type *Route-First Cluster-Second.* Dans la première phase, un « tour géant » est effectué, i.e., par un véhicule de capacité infinie. Ce tour géant est ensuite découpé optimalement en tournées réalisables pour le CARP. Le découpe se fait en calculant un plus court chemin dans un graphe auxiliaire valué H = (X, U, Z). X comprend n + 1 nœuds indexés de 0 à n (n = le nombre de nœuds du graphe original). Tout arc (i, j) de U représente une tournée

réalisable et la valeur z(i,j) sur l'arc est le coût d'une telle tournée. Un plus court chemin entre les nœuds 0 et n dans H fournit un découpage de θ en tournées réalisables minimisant le coût total. Ramdane-Cherif (2002) présente un algorithme pour faire ce découpage dans une procédure Split qui évite une génération explicite du graphe auxiliaire H et minimise en plus le nombre de véhicules en critère secondaire. La procédure est transcrite ci-après :

Algorithme : *SPLIT*

ENTRÉE : un chemin représentant un tour géant d'un graphe G

SORTIE : une solution (pas forcément optimale) du PFC dans G.

```
V(0),N(0) := 0
for i := 1 to n do V(i) := ∞ endfor
for i := 1 to n do
  load, cost := 0; j := i
  repeat
      load := load + r(θ(j))
      if i = j then
            cost := D(s,θ(i)) + w(θ(i)) + D(θ(i),s)
      else
            cost := cost - D(θ(j-1),s) + D(θ(j-1),θ(j)) + w(θ(j)) + D(θ(j),s)
      endif
      if load ≤ Q then
            VNew := V(i-1) + cost
            if (VNew < V(j)) or ((VNew = V(j)) and (N(i-1) + 1 < N(j)) then
                  V(j) := Vnew
                  N(j) := N(i-1) + 1
                  B(j) := i-1
            Endif
            j := j + 1
      endif
  until (j > τ) or (load > Q)
endfor
```

Figure 2.2. Procédure de découpage *Split* pour l'heuristique d'Ulusoy

Parmi les métaheuristiques nous pouvons citer les suivantes :

Méthode Tabou : cette méthode de recherche locale a été utilisée avec d'excellents résultats sur toutes les instances de la littérature par Hertz, et al. (2000). La méthode nommée Carpet consiste d'abord à résoudre un PFR par l'heuristique de Frederickson (1979), puis à construire des solutions du CARP par des procédures de décomposition et manipulation basées sur des voisinages appelés Cut, Switch, Postopt, Shorten, Drop, Add et Drop-Add. Ces procédures peuvent être utilisées comme des outils de base pour la génération d'algorithmes constructifs.

Recuit simulé : cette méthode a été la première métaheuristique proposée pour le CARP, Li (1992), Eglese (1994) et Eglese & Li (1996) ont testé la méthode, ce dernier pour un problème de tournées sur arcs avec plusieurs dépôts pour un problème de sablage de route. Plus récemment Wohlk (2003) a développé des algorithmes de programmation dynamique pour un problème plus général qui sont adaptés au CARP à l'intérieur d'une procédure de recuit simulé.

Recherche à voisinage variable (VNS) : Hertz & Mittaz (2001) ont couplé cette technique avec la méthode Carpet. Les procédures Postopt et Shorten (Hertz, et al., 2000) sont appliquées à chaque voisinage où celui-ci correspond à k tournées, le voisinage est variable à mesure que k varie de 1 jusqu'au nombre de véhicules disponibles. VNS donne des solutions de meilleure qualité que Carpet sur les grandes instances.

Autres algorithmes de recherche locale : Beullens, et al. (2003) utilisent une méthode de recherche locale guidée. La méthode est une variante d'une méthode de descente pour des problèmes de tournées sur les nœuds (Muyldermans, et al., 2001). La méthode réalise deux types des mouvements, des mouvements à l'intérieur de chaque tournée et des mouvements entre tournées. La méthode a une bonne performance.

Algorithmes évolutifs : Lacomme, et al. (2004) ont développé un algorithme génétique qui a donné de très bonnes performances. Une caractéristique importante de cet

algorithme est que la codification de chromosomes n'utilise pas des délimiteurs des tournées. Chaque chromosome représente une grande tournée et une procédure *split* est réalisée après des mouvements (*mutation* ou *exchange*) pour trouver une solution réalisable (voir Figure 2.2).

Formulations linéales et solution exacte : la première formulation par programmation linéaire est dû à Golden & Wong (1981), cette formulation utilise trop de variables, la résolution exacte n'est possible que pour de petites instances. Les principales avancées en termes de modélisation sont dues à Belenguer et Benavent (1992, 1994, 1998) qui ont travaillé sur deux formulations plus compactes, dites *sparse* et *supersparse*. Ils résolvent une formulation relaxée avec une méthode de coupes, obtenant ainsi une excellente borne inférieure. Selon Ghiani & Laporte (2000), la méthode de coupes se présente comme une méthode de grand potentiel. Pour plus de détails sur les différentes formulations, nous recommandons le livre coordonné par Dror (2000). La formulation présentée en Dror & Langevin (2000) est montrée ci-dessous :

Soit le graphe G tel que défini dans la section 2.1, page 10 , et :

- q_{ij} = la demande sur le lien (i,j) $\in L' \subseteq E$;
- V= la borne supérieure du nombre de tournées possibles ;
- Q_v= la capacité de la tournée v ;
- c_{ij} = le coût de traverser le lien (i, j) (initialement non nul) ;
- X_{ijv} = le nombre de fois que le lien (i, j) est traversé dans la tournée v ;
- Y_{ijv} = 1 si le lien (i,j) $\in$ L' est couvert par la tournée v; 0 sinon.

On suppose que le nœud 0 est le dépôt.

Minimiser :

$$\sum_{(i,j)\in E} \sum_{v=1}^{V} c_{ij} X_{ijv}$$

Sujet à :

$$\sum_{k\in N} X_{kiv} - \sum_{k\in N} X_{ikv} = 0, \forall i \in N, v = 1,2...,V \qquad (2.1)$$

$$\sum_{v=1}^{V} X_{ijv} = 1, \forall (i,j) \in L' \qquad (2.2)$$

$$\sum_{(i,j)\in L'} q_{ij} Y_{ijv} \leq Q_v, v = 1,...,v \qquad (2.3)$$

$$X_{ijv} \geq Y_{ijV}, \forall (i,j) \in L', v = 1,...,v \qquad (2.4)$$

$$M \sum_{i\notin N[S], j\notin N[S]} X_{ijv} \geq \sum_{(j,k)\in S} X_{jkv}, \begin{cases} \forall S \subseteq R, \\ 0 \notin N[S], \\ v = 1,...,V \end{cases} \qquad (2.5)$$

$$Y_{ijv} \in \{0,1\}, \forall (i,j) \in L', v = 1,...,V \qquad (2.6)$$

$$X_{ijv} \in Z^{+}, \forall (i,j) \in E, v = 1,...,V \qquad (2.7)$$

où M est une constante de valeur plus grande ou égale au nombre de liens dans tout ensemble S $\subseteq$ R, et N[S] est l'ensemble de nœuds incidents de l'ensemble S.

La fonction-objectif minimise la distance totale parcourue pour toutes les tournées. La première famille de contraintes décrit les contraintes de conservation de flux. La deuxième famille de contraintes oblige à faire le service sur les liens obligatoires exactement une fois. Le troisième groupe de contraintes correspond aux contraintes de capacité. La famille 2.4 de contraintes oblige que la tournée v couvre le lien (i, j), si elle livre la demande. La cinquième famille représente les contraintes d'élimination de sous-cycles, qui assurent que chaque tournée est connectée avec le dépôt.

Des méthodes exactes ont été utilisées pour résoudre le problème. Par exemple Hirabayashi, et al. (1992) et Kiuchi, et al. (1995) ont utilisé la méthode de séparation et évaluation, mais elle est peu performante et elle traite des cas jusqu'à 30 arcs seulement. Plus récemment Belenguer, et al. (2006) ont développé un algorithme de coupes pour le problème général de tournées (GRP), dont le CARP est un cas particulier de ce problème. Dror & Langevin (2000) examinent le problème dans la perspective duale de tournées sur les nœuds. Ils l'ont analysé en faisant une transformation du problème sur

les arcs en un sur les nœuds et proposent une méthode de génération de colonnes pour le résoudre. Plus récemment Baldacci & Maniezoo (2004) examinent aussi le problème dans la perspective duale de tournées sur les nœuds en réalisant une transformation du CARP au VRP et proposent une méthode exacte à partir de la méthode de *Branch and Cut*. Toutefois, ces approches restent à explorer pour des problèmes de grande taille, parce que pour le moment elles ne sont pas résolues en temps acceptables.

2.1.2 Problèmes mixtes de localisation et de tournées sur les sommets

Les problèmes mixtes de localisation et de tournées (PLT) consistent à localiser plusieurs installations qui servent plusieurs clients et trouver l'ensemble optimal de routes pour les visiter. Avec l'intégration de systèmes logistiques pour l'amélioration des processus de distribution, on a vu une évolution pendant les 30 dernières années. La recherche sur PLT est limitée en comparaison avec la littérature étendue sur des problèmes purs de localisation (PL), de tournées de véhicules (PTV), et leurs variantes. Les PTV ont été identifiés en tant que problèmes NP-durs depuis longtemps. Les PLT sont en conséquence plus compliqués. Il n'est pas étonnant que des méthodes d'approximation soient plus largement répandues que des méthodes exactes. En termes de procédure pour résoudre le PLT, il peut être regardé comme la succession de trois sous-problèmes, résolus par étapes. Les étapes principales selon Laporte (1988) sont:

- localisation des installations (L) ;
- affectation des clients à des tournées (A) ;
- génération de tournées (T).

L'optimisation séparée de ces sous-problèmes mène toujours à une décision non optimale. Cependant, et pour le moment, l'incorporation de tous les sous-problèmes ensemble est informatiquement inefficiente en termes de temps d'exécution.

2.1.3 Problèmes mixtes de localisation et de tournées sur les arcs

À la différence du PTA, on ne trouve pas beaucoup de travaux sur le sujet. Ces problèmes se présentent quand on doit traverser quelques arcs et de plus trouver des endroits pour localiser des installations sur le graphe. On trouve des applications pour ce type de problèmes dans le ramassage de courrier, la collecte de déchets et l'enlèvement de la neige. Quelques applications et méthodologies sont présentées par Ghiani & Laporte (2001) et synthétisées dans ce qui suit.

2.1.3.1 Algorithmes exacts

Ghiani & Laporte (1999) ont formulé et résolu de manière exacte le problème mixte de localisation et facteur rural dans un graphe non orienté. Dans le cas, les sites pour localiser les dépôts (D : dépôts potentiels) sont connus au départ et sont un sous-ensemble des sommets (V), c'est-à-dire : $D \subseteq V$. Chaque dépôt potentiel a un coût fixe d'opération et l'objectif dans ce problème est de minimiser le coût pour traverser tous les arcs requis et la somme des coûts d'opération des dépôts. Le problème a été transformé en un problème équivalent de PFR, dans le cas où le nombre de dépôts est illimité, et en un problème relaxé de PFR, dans le cas contraire. Pour le solutionner, ils ont utilisé la méthode de *branch-and-cut* développée par Ghiani & Laporte (2000). Avec cette méthode, ils ont résolu des problèmes de 200 sommets ayant 70% de densité de connexion entre les sommets.

Muyldermans, et al. (2002) ont introduit un problème PLTA dans lequel on doit localiser p installations et minimiser, en même temps, le kilométrage parcouru en itinéraires improductifs, dans un graphe non orienté. Dans cette situation, les véhicules peuvent retourner seulement aux sommets, et la demande peut être divisée. L'auteur montre que si les installations peuvent être localisées partout dans le graphe, l'ensemble de localisations réalisables peut se limiter aux sommets du graphe. Cette proposition est valide dans le cas où les véhicules peuvent retourner partout dans le graphe et les demandes ne peuvent pas être divisées. Selon l'auteur, ces résultats vont dans la même

direction que la propriété d'optimalité de sommets pour le problème de localisation de p installations (*p-median problem*) d'Hakimi (1965). L'article présente des procédures de temps polynomial et pseudo polynomial pour trouver une solution optimale au problème p-PII dans une chaîne non orientée, avec besoin de service sur chaque arête (une chaîne est suivi d'arêtes tel que chaque arête a une extrémité en commun avec la précédente et l'autre avec la suivante).

2.1.3.2 Heuristiques

De la même façon que pour les problèmes de localisation sur les nœuds, la résolution heuristique typique de ce type de problèmes est de résoudre le problème mixte par étape.

Dans le schème donné par Laporte (1988) dans le contexte de localisation et affectation sur les nœuds, on trouve deux situations classiques :

- (L, A, T) : dans ce schème, les installations sont d'abord localisées, après les clients sont affectés aux sites sélectionnés et finalement les tournées sont construites ;
- (A, T, L) : l'ensemble d'arcs (clients) qui appartiennent à un véhicule est déterminé, puis les tournées sont construites et enfin les installations sont localisées.

Dans chaque schème, les étapes peuvent être combinées, par exemple (LA, T), (L, AT), etc.

2.1.3.3 Heuristiques du type (L, A, T)

Dans un contexte de livraison de courrier aux États-Unis, Levy & Bodin (1989) ont développé une heuristique pour adresser un plan de tournées de facteurs. Le problème se réduit à chercher des endroits (points de « parking et livraison ») pour faire la répartition du courrier. Ainsi, le facteur voyage depuis le bureau de poste jusqu'aux points où il débute et termine plusieurs tournées. D'abord, l'algorithme trouve les points de parking

et livraison, puis il fait l'affectation de clients (représentés par des arcs) à ces points et finalement fait les tournées pour ces clients.

2.1.3.4 Heuristiques du type (A, T, L)

Dans un problème d'installation de boîtes de relais le long des routes de facteurs, Bouliane & Laporte (1992) ont utilisé une heuristique du type (AT, L). L'objectif du problème était de minimiser le nombre de sites où seraient placées des boîtes de relais, tout en s'assurant que les facteurs ne transportent jamais un poids excessif de courrier. Pour établir cet objectif une hypothèse fut posée, selon laquelle le coût d'approvisionnement est proportionnel au nombre des boîtes de relais, même s'il est possible de trouver des exemples dans lesquels un nombre plus grand de boîtes peut diminuer le coût de livraison. La solution à ce problème utilise comme entrée les tournées pour la levée des boites à lettres, ensuite le problème est formulé comme un problème de recouvrement d'ensembles (PRE) et a été résolu de façon approximative.

Dans un contexte d'épandage de sel, Lotan, et al. (1996) ont traité le problème mixte de localisation de dépôts supplémentaires (silos) et de tournées des camions. Il ont utilisé une heuristique constructive basée sur une heuristique de deux phases implantés pour le PTAC par Pearn (1991). Dans la première phase de l'algorithme, les tournées se construisent de façon à inclure les liens qui se trouvent très loin du dépôt. Dans cette logique un silo est considéré pour insertion si la capacité requise pour la tournée est égale à deux fois ou plus la capacité du camion. Dans la deuxième phase de l'algorithme, les tournées se construisent pour inclure les liens qui n'ont pas été considérés dans la phase 1. À ce moment, l'algorithme peut ajouter des silos pour diminuer le coût des tournées. Les silos sont localisés aux nœuds du réseau. L'heuristique peut se décrire comme (TA, L).

Li & Eglese (1996) ont considéré un problème de tournées dans une situation d'épandage de sel dans des opérations hivernales. Dans ce problème, la position des réservoirs est connue au départ, la fabrication des tournées est faite avec une heuristique

constructive qui fait chaque tournée en deux phases. La conception de l'heuristique est inspirée du travail de Pearn (1991) pour le PTAC. Le remplissage des camions est considéré dans la construction de chaque tournée. Le camion doit aller à un site de remplissage si la capacité du camion n'est pas suffisante, ou si la quantité de sel qui reste dans le camion est moins que la moitié de sa capacité, et si la distance pour l'épandage est inférieure à la distance de l'endroit de remplissage le plus proche. L'heuristique peut se décrire comme (TA, L).

Dans le problème traité par Ghiani, et al. (2001), les véhicules doivent traverser l'ensemble R d'arcs obligatoires, collecter la demande sur ces arcs et visiter une des installations intermédiaires pour la décharger (PLTA avec II). La capacité des véhicules est limitée. L'article n'a pas pris en compte le problème de localisation des installations intermédiaires. Au contraire, ils ont d'abord défini les endroits (sur les nœuds) ainsi que le dépôt. Ils ont donné deux heuristiques. La première est du type (T, AL) et l'autre du type (TA, L).

La première heuristique est basée sur le problème du PFR : dans une première étape les demandes sur les arcs sont ignorées, l'algorithme trouve la solution optimale du PFR en utilisant l'algorithme de Frederickson, et al. (1978), qui détermine d'abord un arbre de coût minimum de tous les éléments de R et après détermine un couplage de coût minimum sur les sommets de degré impair. Dans une deuxième étape les demandes sont introduites. Les installations intermédiaires sont localisées et les tournées sont construites simultanément. Cela se fait en déterminant une partition optimale de la solution du PFR, selon le modèle « grouper en premier et fabriquer les tournées en second » utilisé par Beasley (1983) pour le problème de tournées de véhicules avec capacité et de Jansen (1993) pour le problème du PTAC.

La deuxième heuristique est basée sur le problème du PTAC. Dans une première étape le problème du PTAC est résolu en utilisant un graphe transformé. Celle-ci se fait en ajoutant des liens entre le dépôt et les installations intermédiaires, avec coûts et demandes nuls. Puis, pour trouver une solution, l'heuristique CARPET de Hertz, et al.

(2000) est utilisée. Finalement, avec des tournées trouvées dans la première étape, l'algorithme ajoute les installations intermédiaires de façon à obtenir une solution réalisable. Ils ont trouvé que la deuxième heuristique produit des meilleurs résultats que la première, mais avec des temps de résolution deux ou trois fois plus longs.

Le même problème des installations intermédiaires, décrit ci-dessus, a été étudié par Ghiani, et al. (2004) et Polacek, et al. (2006). Les méthodes de résolution sont des métaheuristiques, la première avec une méthode tabou et la deuxième avec une méthode de voisinage variable.

De Rosa, et al. (2002) introduisent un problème qui peut être un PLTA si l'on ajoute des décisions de localisation des endroits de transbordement. Dans cette situation, les arcs sont traversés pour cueillir des ressources demandées. Le véhicule va vers une station de transbordement pour laisser le produit qui, là-bas, recevra des traitements spéciaux. Ensuite, une fois vide, le véhicule continue avec la collecte. Dans la station de transbordement, des véhicules de grande capacité transportent du matériel traité au dépôt final.

L'article n'a pas pris en compte le problème de localisation des endroits de transbordement. Au contraire, ils ont d'abord défini les endroits de transbordement ainsi que le dépôt final de destination. Pour résoudre le problème, ils ont utilisé la méthode de recherche taboue. La solution initiale a été conçue en ignorant les demandes et en utilisant la méthode pour le PFR de Corberán & Sanchis (1994). Celle-ci utilise l'heuristique de Frederickson. Finalement le problème d'ordonnancement est traité pour les véhicules de grande taille utilisant une heuristique appelée *go-when-full* (envoyer le véhicule quand il est plein). Le chargement des voitures se fait en accord avec une politique FIFO.

2.1.3.5 Discussion

Le Tableau 2.1 présente un résumé des types des problèmes traités et de la méthode utilisée.

Tableau 2.1. Méthodes de résolution pour le PLTA

	PFC	PFR	PFC-C	PTAC
Non orienté		Guiani, et al. (1999) [a]	Muyldermans, et al. (2002) [a]	Guiani, et al. (2001) [b,c] De Rosa, et al. (2002) [c] Guiani, et al. (2004) [b] Polacek (2006) [b]
Orienté			Bouliane, et al. (1992) [b] Lotan, et al. (1996) [b]	Li, et al. (1996) [b] Levy, et al. (1989) [d]
Mixte				

Types de résolution : [a] : exacte ; [b] : (T,AL) ; [c] : (TA,L) ; [d] : (L,A,T)

La revue de littérature montre que le problème de localisation et construction de tournées sur les arcs a été peu étudié. Nous entrevoyons de nombreuses possibilités de recherche sur ce problème. Quant à la façon de trouver une solution, on peut noter que pour le problème avec capacité, il n'existe présentement des approches exactes que pour le problème où le graphe est une chaîne, ce qui est peu applicable à la réalité. On note aussi que les auteurs ont abordé le problème de localisation des installations sur les nœuds, mais pas quand elles doivent être localisées partout dans le réseau.

Quant aux méthodes heuristiques, Ghiani & Laporte (2001) font ressortir le fait que les heuristiques du type (L, A, T) sont plus faciles à développer, car lors de la dernière étape, on arrive généralement à travailler avec un algorithme polynomial. Mais, la solution de l'étape de construction de tournées est très dépendante de la sélection faite dans les phases d'affectation et de localisation. Cette situation peut produire de très mauvaises solutions. Ils recommandent alors les heuristiques du type (T, A, L).

2.2 Outils pour la modélisation informationnelle

2.2.1 L'approche orientée-objet

L'approche orientée-objet est une méthode permettant de définir un problème et d'aborder sa solution. La modélisation informatique consiste à représenter une partie du monde réel en un ensemble d'entités informatiques. Cette technique peut s'employer comme moyen de modélisation et comme support à la programmation. La façon de travailler avec cette approche se résume en ces trois étapes, tiré de FSC Ltd. (2003) :

- on débute avec une analyse orientée-objet ;
- l'analyse est transformée en un modèle orienté-objet ;
- le modèle-objet est converti en un programme orienté-objet.

Ce qui est différent de l'approche traditionnelle :

- on débute avec une analyse structurée ;
- on développe un dessin modulaire ;
- on écrit les programmes de façon procédurale.

La Figure 2.3 présente un schème comparatif de l'approche structurale et de l'approche orientée-objet, tiré de FSC Ltd. (2003).

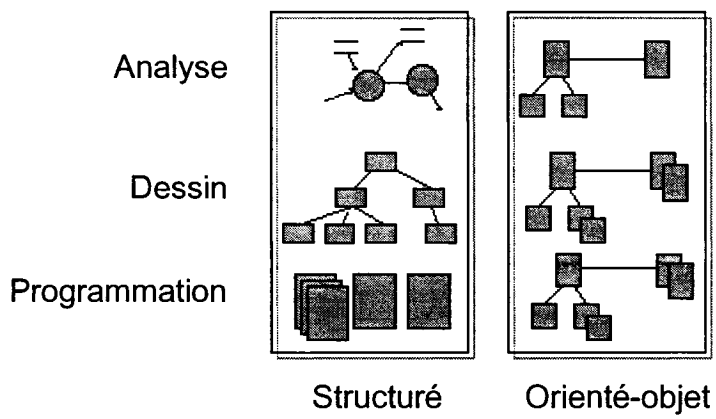


Figure 2.3. L'approche traditionnelle versus l'approche orientée-objet

Trépanier (1999) résume les principaux avantages et inconvénients de l'utilisation de l'approche orientée-objet :

Tableau 2.2. Avantages et inconvénients de l'approche orientée-objet

Avantages	Inconvénients
• Développement plus rapide de logiciels • Qualité supérieure des applications • Entretien aisé des systèmes • Coûts réduits • Mise à l'échelle rapide des logiciels • Meilleur structure informationnelle • Meilleure adaptabilité	• Demande une technologie plus mature • Nécessite des normes d'application strictes • Nécessite des outils de programmation efficace • Vitesse d'exécution plus lente • Nécessite un personnel qualifié • Coûts de conversion des logiciels existants

2.2.2 Éléments de la programmation orientée-objet

2.2.2.1 Objet

Dans la modélisation orientée-objet les entités informatiques sont appelées objets. Un objet est une représentation abstraite d'une chose ayant une existence matérielle (maison, école, véhicule, ...) ou bien virtuelle (tournée, chemin, temps, ...). Un objet est caractérisé par :

- les attributs : ce sont des variables ou données qui caractérisent l'objet ;
- les méthodes : les ensembles de procédures ou opérations que l'objet peut réaliser.

2.2.2.2 Classe

Une clase est un regroupement d'objets de même nature. Un objet est une instanciation d'une classe. Une classe est composée de deux parties :

- les attributs : ce sont des variables ou données qui caractérisant les objets ;
- les méthodes : les ensembles de procédures ou opérations que les objets peuvent réaliser.

2.2.2.3 Encapsulation

L'encapsulation est un mécanisme permettant de réunir des données (attributs) et des procédures (méthodes) dans une même entité, appelée objet, cette structure est indépendante de l'implantation de l'objet. L'encapsulation permet de définir des niveaux de visibilité des éléments de la classe. Ces niveaux sont les suivantes :

- public : il n'y a pas de restriction entre classes aux accès de données ou aux méthodes ;
- protégé : l'accès aux données est réservé aux fonctions membres de la classe et aux classes dérivées ;
- privé : l'accès aux données et méthodes est limité à la classe elle-même.

2.2.2.4 Héritage

C'est une façon de créer des classes spécialisées. L'héritage est la possibilité de créer une nouvelle classe (classe dérivée) à partir d'une classe existante (superclasse). La classe dérivée hérite les attributs et méthodes de la superclasse, la classe dérivée peut elle-même aussi définir de nouveaux attributs et de nouvelles méthodes.

2.2.2.5 Héritage multiple

Possibilité de faire hériter une classe de deux ou plus superclasses. Les attributs et méthodes des superclasses sont dérivés.

2.2.2.6 Polymorphisme

Caractéristique permettant qu'une classe puisse prendre plusieurs formes. On distingue généralement trois types de polymorphisme :

- le polymorphisme surcharge (en anglais *overloading*) : permet d'avoir des fonctions de même nom avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles ;
- le polymorphisme d'héritage (en anglais *overriding*) : permet d'utiliser la méthode d'un objet sans se soucier de son type intrinsèque ;
- le polymorphisme paramétrique (en anglais *template*) : permet de définir plusieurs fonctions de même nom mais possédant des paramètres différents.

2.2.3 L'approche orientée-objet en transport

L'approche orientée-objet a été utilisée par des chercheurs en transport. Il existe différentes techniques et méthodes d'analyse et de modélisation utilisées par les planificateurs de transport. Trépanier (1999) cite quelques unes comme préambule a son propre modèle :

- le projet PoeT de l'université de Leeds (Foster, et al., 1994) ;
- l'outil STROBOSCOPE de l'université du Michigan (Martinez, 1996) ;
- le système d'information objet à la Régie Autonome des Transports Parisiens (RATP) de Rizzi & Guichoux (1997) ;
- le modèle URBAN de Rafanelli (1998).

Le modèle de Trépanier est basé sur les quatre grandes classes d'objets de transport, développées au sein de la modélisation orientée-objet en transport (MOOT) de Trépanier, et al. (2002), c'est-à-dire :

- les objets statiques, qui possèdent une localisation fixe dans le temps et l'espace (codes postaux) ;
- les objets dynamiques, qui sont les acteurs du transport (une personne, un véhicule) ;
- les objets cinétiques, qui sont les descripteurs du mouvement (un chemin, un circuit) ;
- les objets systémiques, qui sont des groupes d'objets interreliés (réseau routier).

La Figure 2.4, tirée de Trépanier (1999), représente ces quatre métaclasses ainsi que la notation utilisée pour caractériser les liens existants entre ces différents objets.

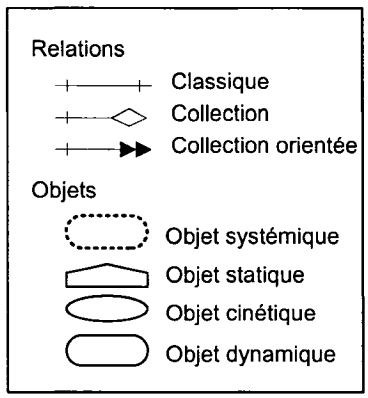


Figure 2.4. Description des objets en transport et caractéristiques des relations

Si l'on s'intéresse à l'utilisation de la géomatique comme moyen de soutien aux applications, les systèmes d'information géographiques pour le transport orienté-objet (SIG-TOO) constituent l'approche actuelle la plus pertinente à la préhension des divers problèmes de transport. L'amalgame de "questionneurs", de "calculateurs" et de "présentateurs" réussit à constituer interactivement une géomatique associée à ce type d'application (Trépanier & Chapleau, 2001).

2.2.4 UML

L'UML (Unified Modeling Language) ou « langage de modélisation unifié » est une notation permettant de modéliser un problème de façon standard. Ce langage provient de la fusion de plusieurs méthodes existant auparavant. UML est un langage pour la modélisation. En résumé, UML est (Popineau, 2000) :

- une notation, pas une méthode ;
- un langage de modélisation objet ;
- le fruit de l'expérience et des besoins de la communauté des utilisateurs ;
- adapté à toutes les méthodes objet ;
- dans le domaine public.

UML permet de définir et de visualiser un modèle à l'aide de diagrammes qui possèdent une structure. Les différents types de diagrammes UML combinés offrent une vue complète des aspects statiques et dynamiques d'un système. UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux vues. Popineau (2000) résume la structure du langage dans le tableau suivante :

Tableau 2.3. La structure de l'UML

Composants	Structuraux	Classe
		interface collaboration cas d'utilisation classe active composant nœud
	comportementaux	interaction machine à états
	regroupement	package
	annotation	note

Tableau 2.3. La structure de l'UML (suite)

Composants	Structuraux	Classe
Relations		dépendance association généralisation réalisation
diagrammes	vues statiques	diagrammes de cas d'utilisation diagrammes d'objets diagrammes de classes diagrammes de composants diagrammes de déploiement
	vues dynamiques	diagrammes de collaboration diagrammes de séquence diagrammes d'états-transitions diagrammes d'activités

2.2.5 Systèmes d'information géographique

Les systèmes d'information géographique (SIG) ont de nombreuses utilisations et applications et par conséquent plusieurs définitions. L'ESRI *(Environmental Systems Research Institute*) les définit comme un outil informationnel pour analyser et visualiser les objets qui se trouvent et les événements qui se produisent sur la terre (ESRI, 2003a). En complément de la définition classique des systèmes d'information, on les définis comme l'ensemble de matériels, logiciels, bases de données, personnes et procédures organisationnelles pour collecter, approvisionner, analyser et communiquer de l'information particulière de la terre (ESRI, 2003b). Les SIG intègrent les opérations communes des bases de données telles que les requêtes et les analyses statistiques avec l'avantage de visualisation et des analyses géographiques offertes par les cartes. Avec l'intégration des SIG et des procédures ou algorithmes pour faire des analyses et solutionner des problèmes de routage et de localisation dans un réseau, on a vu naître une nouvelle gamme d'outils pour l'analyse et la planification du transport. Les systèmes d'information géographique pour le transport (SIG-T; en anglais *Geographic*

Information System for Transportation : GIS-T) ne sont qu'une spécialisation des SIG, utilisées, influencées et destinées aux activités de transport (Fletcher, 2000) et réservées à de nombreuses utilisations et défis (voir Goodchild (2000)).

Au début de l'utilisation des SIG-T, les chercheurs se sont confrontés à des difficultés pour les adapter à des méthodologies de planification du transport. Les articles de Dueker & Butler (1998) et Dueker, et al. (2000) présentent des aspects théoriques et d'implantation de leur initiative pour développer un modèle de données applicable à un processus de transport. Le modèle contient quatre composants :

- un inventaire d'objets de transport comportant : la juridiction, les dispositifs de transport, les points d'événements, les événements linéaires, les événements de point et les intersections ;
- un réseau qui inclut des nœuds, des liens, des traversées et des segments traversés ;
- un système de référence géodésique consistant en des points géodésiques, objets de référence et d'ancrage ;
- une cartographie qui inclut des fonds de carte, des chaînes d'événement linéaires, des segments de ligne, des symboles de point et des points cartographiques.

L'utilisation des SIG-T est de plus en plus répandue, ils sont utilisés dans des axes de recherche différents mais notamment pour la visualisation des solutions d'optimisation de tournées dans le domaine routier. L'apport de la technologie du SIG-T provient de la technique des éléments graphiques qui offre une présentation simplifiée de l'information. Ainsi, et grâce à d'autres techniques telle que la recherche opérationnelle, il est possible de présenter des cartes qui permettraient au gestionnaire d'évaluer la situation et de prendre une décision dynamique.

Un aspect intéressant à considérer en utilisant un SIG est la façon d'intégrer celui-ci aux outils de planification et d'aide à la prise de décisions (SAD). Goodchild (1992) fait une

classification basée sur l'architecture pour l'intégration, où celle-ci s'exprime en termes de proximité ou d'amplitude dans laquelle deux systèmes sont intégrés. La Figure 2.5 présente trois types d'intégration: faible, forte et complète.

Anselin, et al. (1993) classifient l'intégration des systèmes selon la direction de l'intégration, à savoir: intégration en une direction, intégration en deux directions et intégration dynamique (voir la Figure 2.6 tirée de Chulmin (2000)).

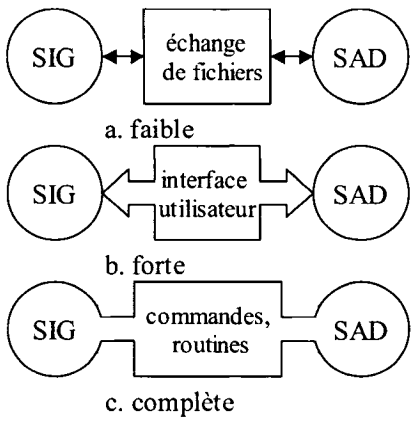


Figure 2.5. Classification basée sur l'architecture pour l'intégration

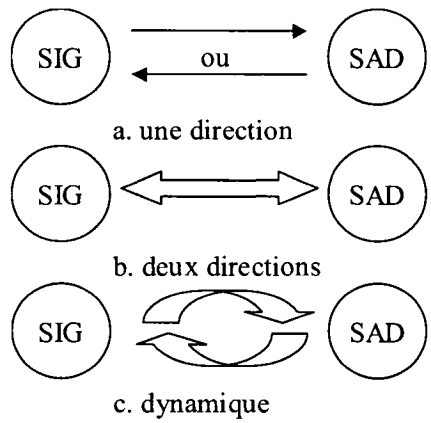


Figure 2.6. Classification selon la direction de l'intégration

Comme Chulmin (2000) l'a aussi souligné, le thème de recherche le plus étudié est l'utilisation des SIG comme fournisseurs de données pour établir les paramètres des modèles et comme visualisation des solutions proposées. Pendant les dernières années, il y a eu une grande augmentation de l'utilisation des SIG. Les applications utilisant les SIG touchent plusieurs domaines du transport.

Le Tableau 2.4 présente des exemples qui concernent l'intégration de tournées de véhicules et de SIG-T.

Tableau 2.4. Intégration des PTV et SIG-T

Auteur	Type de problème	Architecture de l'intégration	Direction de l'intégration	Méthode d'optimisation employée
Wolfler-Calvo, et al. (2004)	Covoiturage (VRP)	forte	une direction	heuristique constructive. Recherche locale
Zaman, et al. (2002)	VRP et ordonnancement	forte	une direction	algorithmes génétiques
Campbell & Langevin (1995)	VRP : opérations de déneigement	forte	une direction	heuristique
Salim, et al. (2002b)	entretien de routes et ponts	forte	une direction	module d'optimisation
Salim, et al. (2002a)	routage dans un système de gestion de ramassage de la neige	forte	une direction	heuristique constructive
Tarantilis, et al. (2002)	PTV	complète	deux directions	métaheuristiques de recherche locale
Amaya, et al. (2003)	VRP- transport d'écoliers	complète	deux directions	heuristique constructive. optimisation locale

Voici quelques points clés au cours de l'intégration de GIS et des outils de planification :

- les chercheurs deviennent intéressés au développement d'un module intégré de décision à l'intérieur du GIS, pour pouvoir exécuter les outils dans un environnement plus flexible et plus convivial ;
- les SIG sont des outils très appréciés quand le type de problème exige l'analyse des facteurs qui dépendent de l'information des lieux géographiques et quand les planificateurs doivent prendre des décisions multicritères, et ce avec différents niveaux de priorité (Chulmin (2000) ;
- les chercheurs ont des problèmes à intégrer les données opérationnelles à celles de planification et de géo-référence ;
- problèmes de représentation de données temporelles.

CHAPITRE 3 : THE CAPACITATED ARC ROUTING PROBLEM WITH REFILL POINTS

Article écrit par Alberto Amaya, André Langevin et Martin Trépanier (Amaya, et al., 2007) ; accepté pour publication dans Operations Research Letters. Disponible en ligne : Doi:10.1016/j.orl.2005.12.009

Dans l'étape de planification saisonnière, nous avons identifié un problème qui est une variante des problèmes mixtes de localisation et de tournées sur les arcs (PLTA) que nous appelons le "problème de tournées sur les arcs avec points d'arrêt pour remplissage". On trouve ce problème dans les opérations d'entretien des chaussées et spécifiquement dans les opérations de marquage du réseau routier. Dans ce problème, le planificateur doit élaborer les circuits qu'un véhicule spécialisé doit parcourir afin de tracer les lignes de signalisation sur la chaussée. Ce véhicule, appelé véhicule traceur de lignes, a un réservoir de peinture de capacité limitée, donc il doit être rempli régulièrement pour continuer son parcours de traçage. Dans la situation étudiée, le véhicule traceur de lignes peut être rempli sur place par un autre véhicule appelé le véhicule citerne. Le problème consiste à déterminer simultanément les tournées à suivre par les deux véhicules de façon à ce que le coût total soit minimisé.

Nous avons étudié différentes contraintes caractéristiques de ce problème, mais en particulier, dans ce chapitre, nous étudions une contrainte du véhicule citerne dans laquelle il est obligé de retourner au dépôt chaque fois qu'il rencontre le véhicule traceur de lignes. Cette situation est caractéristique des opérations actuelles dans la province de Québec. Cette contrainte sera éliminée dans un deuxième modèle qui sera présenté dans le chapitre suivant.

L'article développe un modèle de programmation linéaire en nombres entiers. Le modèle mathématique est bâti sur un graphe mixte. La formulation est inspirée des modèles du CARP (*cacitated arc routing problem*). Une formulation du type *sparse* est utilisée, c'est–à-dire le nombre de variables est proportionnel au nombre d'arêtes et arcs dans le graphe. L'ensemble des arêtes est transformé en arcs en remplaçant chaque arête par deux arcs opposés et en ajoutant une contrainte (famille de contraintes 4) pour restreindre le passage de service une seule fois dans l'une des deux directions. Pour résoudre le problème, le graphe a été augmenté en ajoutant deux ensembles de liens qui connectent chaque nœud avec le dépôt et le dépôt avec chaque nœud (voir section 3.3.1).

Deux types des variables ont été utilisés :

- x_{pa} = le nombre de fois que chaque arc a été visité entre le ravitaillement p et le ravitaillement p+1 et
- y_{pa} = 1 si l'arc est à desservir entre le ravitaillement p et le ravitaillement p+1; 0 sinon.

Dans ce problème on a considéré trois types de coûts : les coûts de traverser et de servir un arc par le véhicule traceur de lignes et le coût de traverser un arc pour le véhicule citerne. Dans la fonction-objectif on n'a pas inclus le coût de servir un arc par le véhicule traceur de lignes parce que ce coût n'a pas d'effet dans l'optimisation du problème. L'ensemble complet de contraintes est présenté à la section 3.4.

Pour tester le modèle et la formulation proposée, un algorithme de coupes a été développé. Cet algorithme fait une relaxation des contraintes de connectivité (famille de contraintes 11). À chaque itération, les contraintes qui ont été violées sont ajoutées jusqu'à obtenir une solution réalisable (voir section 3.5).

On a trouvé que la méthode proposée est capable de résoudre en temps acceptable des problèmes de petite taille pour l'ensemble de problèmes définis sur des graphes mixtes et des problèmes de petite et moyenne tailles pour l'ensemble des problèmes sur des graphes orientés.

Dans cet article, on a réussi à présenter un nouveau problème de tournées sur les arcs et à le modéliser en utilisant la programmation mathématique en nombres entiers. On a aussi réussi à résoudre des problèmes test de petite et moyenne tailles avec une méthode de coupes.

Malheureusement les problèmes réels sont des problèmes de grand taille qu'on n'est pas capable des résoudre avec cette méthodologie en un temps raisonnable. Pour cette raison, le chapitre suivant présente un modèle amélioré et une méthode de résolution heuristique qui permettra de résoudre des problèmes de grande taille comme ceux du MTQ.

Ci-après, le papier soumis et accepté dans la revue *Operations Research Letters*.

The Capacitated Arc Routing Problem with Refill Points

Alberto Amaya, André Langevin, Martin Trépanier

CRT, GERAD, and Department of Mathematics and Industrial Engineering

École Polytechnique de Montréal

C.P. 6079, Succ. Centre-ville, Montréal, Québec, Canada. H3C 3A7

3.1 Abstract

The Capacitated Arc Routing Problem with Refill Points (CARP-RP) is a new variant of the Capacitated Arc Routing Problem (CARP). In a CARP situation, the vehicle servicing arcs has a finite capacity, and hence has to return to the depot or a secondary station to be refilled (or emptied). In contrast, in the CARP-RP, the vehicle servicing arcs must be refilled on the spot by using a second vehicle. The problem consists on simultaneously determining the vehicles routes that minimize the total cost. An integer linear programming model is proposed and computational experiments are presented.

Keywords: Multi-trip rural postman problem; Capacitated arc routing problem; Replenishment points; Integer linear model; Road marking.

3.2 Introduction

The aim of this paper is to introduce the Capacitated Arc Routing Problem with Refill Points (CARP-RP). A practical application of this problem can be found in road network maintenance where the road markings have to be painted or repainted every year, as is the case in the Province of Quebec, Canada. The Quebec Ministry of Transport (MTQ) uses a fleet of special vehicles to mark the roads and also tank trucks to meet the marking vehicle and replenish them. There are special operational conditions that force the tank truck to return to the depot each time it meets the marking vehicle.

The CARP-RP is a variant of the Capacitated Arc Routing Problem (CARP) introduced by Golden & Wong (1981). In the classical CARP (or Multi-trip Rural Postman Problem) a set of minimal cost routes for a vehicle of finite capacity is looked for. The vehicle, usually a fleet of identical vehicles, is based at a depot and each trip starts and ends there. A fleet must service a subset of required edges, on a connected and (usually) undirected graph, and collect (or deliver) the associated quantities, without exceeding the capacity.

Different mathematical formulations have been proposed for the CARP (Golden & Wong, 1981 ; Belenguer & Benavent, 1998 ; Eglese & Letchford, 2000), all of them for the undirected case. The problem we deal with is however characterized by the fact that the road network must be modeled as a mixed graph. In the process of marking the roads, the central line and lane separators must be painted. For the first one, we can traverse the road in any direction, but in the second case it must be done in the same direction as the road lane. Thus the problem to solve is on a mixed graph.

In the CARP-RP, we deal with two different types of vehicles: the first type is used to service the arcs and the other to replenish the first type of vehicle. The vehicle of the first type (called servicing vehicle - SV), which has a finite capacity, and the vehicle of the second type (called refilling vehicle - RV) can meet at any place in order for the first one to be refilled and to continue its service.

The CARP-RP can be viewed as a Location-Arc Routing Problem (LARP) because in addition to traversing edges or arcs, the refill points must also be located on the graph. Many articles have been written on *pure* Arc Routing Problems (ARP) but relatively few studies address location decisions. Ghiani & Laporte (2001) present the LARPs as extensions of one of three classical ARPs: the Chinese Postman Problem (CPP), the Rural Postman Problem (RPP), and the Capacitated Arc Routing Problem (CARP). In most cases the LARPs are solved using heuristics that use a decomposition of the problem into its main components: facility location (L), allocation of vertices to vehicle routes (A), and routing (R). For example, Levy & Bodin (1989) in a mail delivery

context use the following method: first locate facilities, then allocate users, and finally define the routes (an L-A-R scheme). There are also R-A-L schemes such as the one used by Ghiani, et al. (2001) where they introduce a variant of the classical CARP in which the vehicle, starting from depot, traverses edges to collect demands, but instead of returning to the depot when filled up, visits intermediate facilities to unload. They present two heuristics. In the first approach, an R-A-L scheme, an RPP solution is first determined, then demands are reintroduced and the intermediate facilities are inserted by means of a shortest path algorithm and finally the route is split into several routes separated by facilities. In the second approach, an R-A-L scheme, similar to Lotan, et al. (1996), a CARP solution is found in a first step and then facilities are added in order to make it feasible for the LARP.

An exact approach for the LARP is found in Ghiani & Laporte (1999), where they investigate an undirected Location Rural Postman Problem in which the aim is to determine a set of depots among a set of potential ones, and a route for each depot in such a way that the total cost is minimized. The problem is reduced to a RPP if there are no bounds on the number of depots to be opened or to a RPP relaxation otherwise. The problem was solved applying the exact branch-and-cut procedure developed by Ghiani & Laporte (2000) for the undirected RPP. Note that in the previous mentioned problems, there is only "one type" of vehicle. The cost depends on the arcs traversed by this (these) vehicle(s). In contrast, the CARP-RP utilises two different types of vehicles, and the total cost depends on the arcs traversed by them.

The CARP-RP is NP-hard since it reduces to the Rural Postman Problem (RPP), another NP-hard problem, when the capacity of SV is infinite. Moreover, the CARP-RP corresponds to the CARP when the costs incurred by the RV on traversing arcs are greater than the SV costs, since it is then always better for the SV to return to the depot to be replenished.

This paper presents an integer linear programming model for the Capacitated Arc Routing Problem with Refill Points and a cutting plane method to solve it.

The paper is organized as follows. Section 3.3 presents the mathematical formulation for the CARP-RP. Computational results are presented in Section 3.5 and the conclusions follow in Section 3.6.

3.3 Mathematical formulation

The CARP-RP is defined on a mixed graph $G_1 = (N; A_1 \cup E)$, where $N = \{n_1, \ldots, n_{|N|}\}$ is a set of nodes, including a depot n_{depot}, A_1 is a set of directed arcs $\{a_{ij} \mid a_{ij} = (n_i, n_j)$ and $i \neq j\}$, including a subset R_1 of required arcs, and E is a set of edges $\{e_{ij} \mid e_{ij} = (n_i, n_j)$ and $i \neq j\}$, including a subset R_2 of required edges. Each arc of R_1 and each edge of R_2 must be serviced once but they can be traversed several times. With each arc (or edge) a are associated two traversal nonnegative cost:

- c_a: cost incurred by the SV on traversing an arc $a \in A$, $(c_a \geq 0)$
- b_a: cost incurred by the RV on traversing an arc $a \in$ A, $(b_a \geq 0)$

and a nonnegative demand q_a. If $a \notin R_1 \cup R_2$, then $q_a = 0$.

The objective is to determine simultaneously the vehicle route ($n_{i1} = n_{depot}, n_{i2}, \ldots, n_{it} = n_{depot}$) for the servicing vehicle (SV) and the circuits ($n_{i1} = n_{depot}, \ldots, n_{ij}, \ldots n_{ik} = n_{depot}$) for the refilling vehicle (RV) that minimize the total costs while the total demand of any SV path between two consecutives refill points does not exceed the SV capacity.

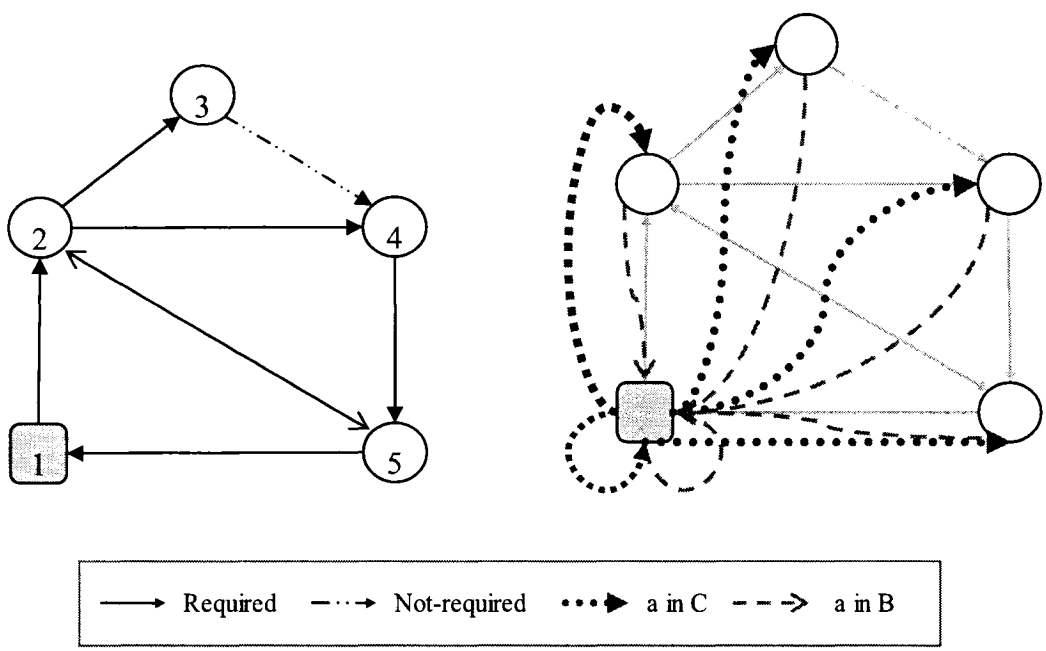
We have used a sparse formulation; i.e., the number of variables is proportional to the number of arcs and edges for each refill (see Eglese & Letchford (2000)). The initial graph G_1 is transformed by replacing each edge by two arcs, with same costs as the edge has. All such arcs are included into A_1 to yield an extended arc set A. Constraints in the model ensure that one of the two arcs is serviced. Similarly, if an edge is required then the two arcs generated are included in R_1 to yield R. At the end of this transformation we have a directed graph. Over this transformed graph we have added two sets of dummy arcs, which are described in the next part of this section.

3.3.1 Graph construction

To deal with the problem, we construct an auxiliary directed graph $\boldsymbol{H}$ = (N, L= ($A \cup B \cup C$)), where:

- $\boldsymbol{A}$ = $\{a_1, \ldots, a_{|A1|+2*|E|}\}$: set of arcs after each edge was coded as two opposite arcs. In this case we have $|R| = |R_1| + 2|R_2|$, where $\boldsymbol{R} \subseteq \boldsymbol{A}$ is the set of required arcs
- $\boldsymbol{B}$ = $\{a_{|A1|+2|E|+1}, \ldots, a_{|A1|+2|E|+|N|} \mid a_{|A1|+2|E|+i} = (n_i, n_{depot})\}$: set of "artificial arcs" connecting each node $n \in N$ to the depot node. And define c_a and q_a, $\forall a \in \boldsymbol{B}$, as follows:

  $\boldsymbol{c_a}$ = the shortest path cost of a *round* trip of the RV, from the depot to the corresponding node n_i of N, computed with the costs b_a
- $\boldsymbol{q_a} = 0$
- $\boldsymbol{C}$ = $\{ a_{|A1|+2|E|+|N|+1}, \ldots, a_{|A1|+2*|E|+2|N|} \mid a_{|A1|+2|E|+|N|+i} = (n_{depot}, n_i)\}$: set of "artificial arcs" connecting the depot with each node $n \in \boldsymbol{N}$. With $c_a = 0$ and $q_i = 0$, $\forall a \in C$



a) Original graph (G_1), with directed and undirected links, and required links

b) Auxiliary directed graph (H)

Figure 3.1. Graph transformation

Figure 3.1 illustrates the auxiliary graph constructed from the original graph. Note that H is not necessarily a simple graph.

We also define the following:

- ***O(n)*** $\subseteq L$: the set of arcs leaving node n, $n \in N$
- ***I(n)*** $\subseteq L$: the set of arcs entering node n, $n \in N$
- ***N(S)*** $\subseteq N$: set of incidents nodes of the arcs of a subset $S \subseteq L$
- ***I(N(S))*** or reduced ***I(S)*** $=\{a \mid a \in \bigcup_{n \in N(S)} I(n)$ but $a \notin S\}$: the set of arcs entering the set $S \subseteq L$
- ***ArtB(n)*** $= B \cap O(n)$: arc of B leaving from node n
- ***ArtC(n)*** $= C \cap I(n)$: arc of C entering to node n
- ***Q*** : the capacity of SV
- ***P*** : the number of times the SV will be refilled (including the first filling at the depot). In this paper we limit this number by a positive integer

And finally let:

- ***Opp(a)*** = the index of the opposite arc. If two arcs a and b represent the same edge, then $Opp(a) = b$ and $Opp(b) = a$. If a is a genuine arc ($a \in A_1$), then $Opp(a) = 0$
- ***W(H)*** $= \{S \subseteq A \mid n_{depot} \notin N(S)$, $S \cap R \neq \varnothing$, $H''=(N(S), S)$ is strongly connected$\}$: the set of arcs that induce strongly connected sub-graphs H''

Note that the depot node is designed as n_{dcpot} or simply *depot*.

3.3.2 Decisional variables

Let

x_{pa} = the number of times arc $a \in L=(A \cup B \cup C)$ is traversed after refill p and before refill $p+1$ and

y_{pa} = 1 if arc $a \in R$ is serviced after refill p and before refill $p+1$; 0 otherwise.

We assume that vehicle starts at the depot with full capacity and this is the first refill. The variables y needs to be defined only for the required arcs. The total number of variables is then $P^*(|A|+2^*|N|+|R|)$.

3.4 Formulation

Minimize:

$$\sum_{p=1..P}\sum_{a\in L} c_a x_{pa} \tag{3.1}$$

Subject to:

$\forall p = 1...P, \forall n \in N$: $$\sum_{a\in I(n)} x_{pa} - \sum_{a\in O(n)} x_{pa} = 0 \tag{3.2}$$

$\forall a \in R, Opp(a)=0$: $$\sum_{p=1..P} y_{pa} = 1 \tag{3.3}$$

$\forall a \in R, Opp(a)>0$: $$\sum_{p=1..P} \left(y_{pa} + y_{p,Opp(a)}\right) = 1 \tag{3.4}$$

$\forall p = 1...P, \forall a \in R$: $$x_{pa} >= y_{pa} \tag{3.5}$$

$\forall p = 1...P$: $$\sum_{a\in B} x_{pa} = 1 \tag{3.6}$$

$\forall p = 1...P$: $$\sum_{a\in C} x_{pa} = 1 \tag{3.7}$$

$\forall p = 1...P$: $$\sum_{a\in R} q_a y_{pa} \le Q \tag{3.8}$$

$\forall p = 2...P, \forall n \in N$: $$x_{p,ArtC(n)} = x_{p-1,ArtB(n)} \tag{3.9}$$

$$x_{1,ArtC(depot)} = x_{p,ArtB(depot)} = 1 \tag{3.10}$$

$\forall S \in W(H), p = 1...P$; $b \in S$: $$\sum_{a\in I(S)} x_{pa} \ge (x_{pb} / M) \tag{3.11}$$

$\forall p = 1...P$: $$y_{pa} \in \{0,1\} \forall a \in R \;;\; x_{pa} \in Z \;\forall a \in A \;;\; x_{pa} \in \{0,1\} \forall a \in B \cup C \tag{3.12}$$

In this problem we have three main costs, the cost incurred by the SV on servicing required arcs, the cost (distance) incurred by the SV on traversing arcs, and the cost incurred by the RV on traversing arcs of the graph. The first cost does not have any role in solving the CARP-RP as it is a constant because all required arc must be serviced (constraints 3.3 and 3.4). The objective function (3.1) represents the total cost incurred by the SV and RV for traversing arcs. Note that the total distance covered by the RV is obtained by summing the c_a ($a \in B$) of the arcs used (for both servicing and traversal) in the optimal solution. The first set of constraints (3.2) is the classic flow conservation constraints for network flow formulations. The set of constraints (3.5) ensure that an arc serviced is necessarily traversed. The set of constraints (3.6) and (3.7) requires that only one artificial arc must be traversed on each refill. The set (3.8) defines the SV autonomy constraints. The set of constraints (3.9) requires that tour k begins where tour (k-1) finishes. The set of constraint (3.10) forces to begin and finish the SV tour at the depot. Constraints (3.11) are the connectivity constraints. M is a number equal to or greater than the maximum number of times an arc will be traversed. If an arc in S is traversed, at least one entering arc to set S must be traversed. The connectivity constraints could be strengthened by substituting the right-hand side "x_{pb}/M" by "y_{pb}" $\forall b \in S \cap R$. However, this substitution is not usable in the strategy of solution we implemented. Finally, constraints (3.12) define the variables.

Note that we can not transform the graph as proposed by Christofides, et al. (1986) who worked on a transformed graph where only the vertices in V_R (the set of the end vertices of the edges in R) appeared. Such transformed graph is constructed by adding an edge between every pair of vertices of V_R and associating to it a cost equal to the shortest path in the initial graph. By deleting vertices (arcs in our case), we remove the possibility of refill at a node other than those of the required arcs.

When the arc traversing costs are equal for the SV and the RV ($c_a = b_a$, $a \in A$) the total optimal cost obtained in the solution for the CARP-RP is also an optimal cost for the equivalent CARP (or Multi-trip Rural Postman Problem). This is because it is

equivalent in terms of cost that when the SV becomes empty, it travels to the depot, gets replenished and returns, or that the RV exits from the depot, meets the SV and returns.

3.5 Computational experiments

In a real problem, like that of marking roads, we deal with a large network. As the CARP-RP is NP-hard, only small problems (not real problems) can be solved to optimality. In this paper we present a new problem and we are mainly interested in the formulation, so we have used to test the formulation a classical cutting plane approach similar to that of Lacomme, et al. (2002). This method is easy to implement. More efficient (and complex) methods like branch-and-cut could be used but with a longer implementation time.

Basic procedure: The cutting plane approach consists in omitting the connectivity constraints of the original problem (P), since not all connectivity inequalities are active in the optimal solution, and trying to solve this new problem (P'). By means of an iterative process, the constraint set of P' is updated by adding the violated constraints of P in the optimal solution of P'. The process goes on until the optimum found for P' is feasible for P. A cutting plane algorithm is expected to reach the optimal solution using only a partial description of P. However, in the worst case, it can perform a very large number of iterations as there are an exponential number of connectivity constraints.

In this kind of problems, as a lot of sub-tours can exist, a large number of connectivity constraints must be added before an optimal solution can be found. To obtain lower and upper bounds quickly, a strategy in two phases described next was implemented.

Phase 1: Finding connectivity constraints (sub-tours)

The number of the branch and bound nodes is limited to 5000 for each iteration of the basic procedure (i.e., solving P'). At this phase, a lower bound is found if the iteration finishes before visiting the 5000 nodes. Of course, if it's feasible for the original

problem, an optimum solution would be found. An upper bound is found if at the end of the iteration (after 5000 nodes) the solution found is feasible for the original problem.

Phase 2: Finding an upper bound

In the basic procedure, after each iteration we add the violated constraints. This means that separated connected components that do not include the depot are identified and added as connectivity constraints. In this phase, we added only the components that include the depot. This process permits to find a solution (not necessarily the optimal solution) in a few iterations. Of course, if the solution found with this method is equal to a known lower bound, the solution is optimal.

To explain why we can not guaranty the optimality, consider a graph with three components $\{C_1, C_2, \text{and } C3\}$. Suppose the optimal solution is to service the C_1 component, which includes the depot, with the first refill and the others with the second refill. Now suppose we have an intermediate solution with C_1 and C_3 with the first refill and C_2 with the second refill. Suppose the intermediate solution is not valid because C1 and C3 are disconnected (connectivity constraint violation). If we add the violated constraint corresponding to the component that includes the depot (component C1), we eliminate the optimal solution.

The model and routine that identify separated connected components were programmed using OPL studio 3.6 running with Cplex 8.0 on a 700 MHz Pentium 3 PC.

3.5.1 A numerical example

We first illustrate the problem with a numerical example. Consider the following graph:

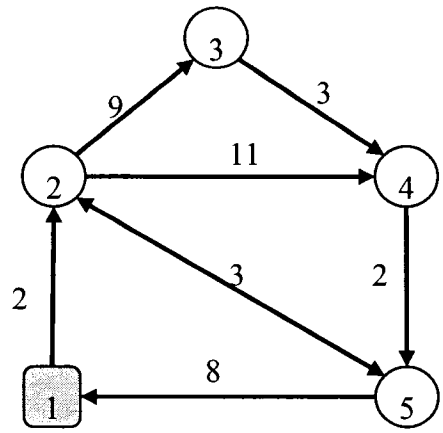


Figure 3.2. Numerical example

Let $c_a = b_a = q_a$ correspond to the number near each arc in Figure 3.2. In this example all arcs are required ($R=A$), the capacity of SV is 15. As the total required length is 41, we can set P *as* 3. Node 1 is the *depot*.

Using Dijkstra's algorithm we can find the associated costs of the set B: $\{c_{a\in B}\}=\{c_{1,1}=0$, $c_{2,1}=13$, $c_{3,1}=24$, $c_{4,1}=23$, $c_{5,1}=13\}$.

Using the basic procedure mentioned earlier, the optimal solution found is:

{ (1,2),(2,4),(4,5), *rf* ,(5,2),(2,3),(3,4)(4,5),(5,1), *rf* ,(1,2),(2,5),(5,1) }

The underlined arcs are traversed but not serviced. The symbol *rf* represents a refill node. So the SV is initially replenished at node 1 (depot), then at node 5 and again at the depot.

The total cost = 66 (traversal cost of SV = 53, traversal cost of RV = $c_{5,1}$= 13).

At the first iteration, the solution of the problem (without any connectivity constraints) was not feasible. Adding the violated connectivity constraint allowed us to find the solution depicted here.

Note that in this problem, the tour done by SV is not an optimal Postman tour. The optimal Postman tour cost would be 46, for example:

$$(1,2),(2,4),(4,5),(5,2),(2,3),(3,4)(4,5)(5,2)(\ 2,5),(5,1)\ .$$

In this case, the RV must be refilled at nodes 4 and 5 with a cost of 36, giving a total cost of 82, which is greater than the optimal cost already found. And it is the same for all optimal postman tours. So this problem can not be solved optimally by finding all postman tours and finding the one that minimizes the RV traversing cost.

3.5.2 Computational results

In order to test the model we used two sets of problems. In the first set of problems, we are interested in oriented networks. In Quebec, for road marking, the central line and the side line can be painted at the same time, and it must be done in the same direction as the flow of traffic, so an appropriate representation of the road network is a directed graph.

We generated different types of problems using a procedure similar to that of Hertz, et al. (2000), all of them on a directed graph.

- We generated randomly the graphs with 3 different types of intervals for the number of nodes ([20, 30], [40, 50], and [60, 70]). The node coordinates were generated in the $[0,300]^2$ square, using a uniform random distribution.
- We generated 2 types of intervals for the number of arcs between [2.5, 4.5] and [6.5, 8.5] times the number of nodes. Each arc of a random Hamiltonian tour was added to ensure the graph connectivity. The prescribed number of arcs was completed by randomly generating the two incident vertices, without repeating arcs.
- The arc demands were generated using a uniform random number between [1, 40].

- The proportion of required arcs (card(R)) was generated using a uniform random number between [70%, 90%], and were designated randomly.
- The cost of traversing arcs (c_a) was set as the length of each arc (Euclidean distance between tail and head nodes), rounded to the nearest integer.
- The cost of traversing arcs incurred by the RV (b_a) was set as a fraction of c_a using a random uniform number between [0.7, 0.9], rounded to the nearest integer.
- The capacity of each SV was set in order to have 3, 4 and 5 replenishments for each set of problems.

For each parameter combination (type of number of nodes, type of number or arcs, type of capacity) we generated 10 problems, so combining all parameters gives 180 instances. These instances contain between 20 and 70 nodes, between 50 and 595 arcs, and between 70% and 90% of required edges.

For this set of problems we used only the Phase 2, described earlier. The computational results are presented in Table 3.1. The column headings are as follows:

- N: number of nodes
- A: number of arcs
- RF: number of refills
- time, av. iter: average time in each iteration (in seconds)
- time, av. tot: average total time of procedure (in seconds)
- time, max tot: maximum time in an iteration to find an optimal solution
- iterations, av: average number of iterations to find the optimal solution
- iteration, max: maximum number of iterations to find an optimal solution
- #opts: the number of problems (out of 10) were the solution was proved optimal
- max gap: maximum difference between the upper and lower bounds (in percentage)

The computational results on these randomly generated problems indicate that the proposed approach can be used to solve to optimality small size problems. The instances solved contain between 20 and 70 nodes and between 50 and 595 arcs. For larger problems computation times become rather large.

Table 3.1. Numerical results for the randomly generated problems

			time			iterations			
N	A / N	RF	av. iter	av. tot	max tot	av	max	# opts	max gap
[20,30]	[2.5 , 4.5]	3	1.4	2.3	2.2	1.5	4	9	1.7%
		4	3.1	12.9	7.7	3.9	10	7	2.9%
		5	6.7	51.0	20.0	6.8	11	7	4.0%
	[6.5 , 8.5]	3	5.5	5.5	11.1	1.0	1	10	0.0%
		4	16.5	27.9	41.1	1.9	4	9	0.1%
		5	35.8	84.1	92.5	2.7	5	9	0.4%
[40,50]	[2.5 , 4.5]	3	2.6	4.7	5.7	1.7	3	9	0.9%
		4	7.6	37.3	26.8	6.1	25	7	1.8%
		5	16.6	299.6	34.9	20.2	106	4	2.8%
	[6.5 , 8.5]	3	13.7	16.3	75.3	1.6	4	10	0.0%
		4	37.1	47.7	101.3	1.3	3	10	0.0%
		5	70.0	155.5	137.4	2.2	4	9	0.1%
[60,70]	[2.5 , 4.5]	3	5.6	11.8	12.1	2.1	7	9	1.4%
		4	11.6	39.1	49.4	3.4	7	8	2.2%
		5	34.4	232.2	78.0	9.1	35	7	3.1%
	[6.5 , 8.5]	3	25.9	25.9	73.4	1.0	1	10	0.0%
		4	115.8	123.2	196.8	1.1	2	10	0.0%
		5	165.3	334.0	308.3	2.2	7	10	0.0%

To further test the formulation, a second set of problem instances was used. They were generated from the classical instances for the CARP introduced by Benavent, et al. (1992); these are undirected CARP instances which give the servicing and the traversing costs for each arc and specify the node corresponding to the depot and the vehicles capacity. We transformed them into mixed CARP-RP instances with the same order of edges as in Belenguer (2006):

- edge no. *i* is transformed into a directed arc, if *(i mod 4) = 1*

- edge no. i is set as not required, if $(i \bmod 4) = 0$, all other edges and arcs are required
- b_a is set as the length
- c_a is set as the weight

The set contains 34 instances which are defined on 10 different graphs. For each graph X, several instances were generated (denoted by X.A, X.B, etc.) by changing the capacity of the vehicles. We used the X.A and X.B sets as they correspond to problems with at most 5 refills. We did not test our methodology for problems with more than 5 refills because the solution time exploded. The resulting problem sets (transformed in mixed graph as explained before and then converted into a completely directed graph as explained in Section 3.3.1) have between 107 and 269 arcs and between 24 and 50 nodes.

Using the basic procedure, described earlier, and limiting the number of iterations (iteration consists in solving the formulation and adding the corresponding connectivity constraints) to 20, we were able to solve 6 problems to optimality. Those problems are shown in bold in Table 3.2.

When using only the basic procedure, we noticed that in most cases the solver spends most of the solution time proving optimality. And in the case where there are no feasible solution for the original problem, and in consequence we would need another iteration, we have to wait some time to discover it. So in this set of problems we used Phase 1 and if no solution was found after 20 iterations, we continued with Phase 2. In phase 1, most of the times, between 1 to 3 connectivity constraints per refill were added so, in average 13 connectivity constraints per iteration.

Computational results are presented in Table 3.2. This table presents for each instance: the number of nodes (N), the number of links after transformation (L), the number of refills (NR), the best lower bound found for each instance (LB), the upper bound (UB),

the number of connectivity constraints added (cuts) at each phase, the number of iterations (iter) at each phase and the CPU time.

The Benavent problems are harder to solve. The problems are on mixed graphs with a lot of edges, thus a large number of cycles can be found. That is different from the first set of randomly generated problems where the graphs are directed and there are few circuits. Real problems would probably contain a lot of circuits separated from the depot as each segment road will be substituted by two opposite arcs (which is not the case with the first set of problems).

Table 3.2. Numerical results for the Benavent set of problems

						Phase 1			Phase 2		
Instance	N	L	NR	LB	UB	cuts	iter	time	cuts	iter	time
tra1A	24	116	2	**405**	**405**	22	6	7	-	-	-
tra1B	24	116	3	**405**	**405**	39	7	21	-	-	-
tra2A	24	107	2	**385**	**385**	90	15	46	-	-	-
tra2B	24	107	3	370	390	138	18	181	-	-	-
tra3A	24	109	2	**164**	**164**	110	14	51	-	-	-
tra3B	24	109	3	**166**	**166**	138	19	140	-	-	-
tra4A	41	202	3	618	640	243	20	300	7	4	129
tra4B	41	202	4	618	690	388	20	752	66	33	1845
tra5A	34	181	3	618	622	147	20	132	2	2	194
tra5B	34	181	4	618	658	280	20	656	40	21	893
tra6A	31	149	3	**483**	**483**	60	11	52	-	-	-
tra6B	31	149	4	477	499	260	17	631	-	-	-
tra7A	40	195	3	634	636	183	13	340	-	-	-
tra7B	40	195	4	634	659	338	20	1145	33	16	1101
tra8A	30	170	3	535	549	135	20	177	11	7	94
tra8B	30	170	4	535	566	284	20	634	35	20	985
tra9A	50	261	3	675	681	186	20	182	7	6	48
tra9B	50	261	4	675	749	368	20	811	7	9	1048
tra10A	50	269	3	670	689	261	20	529	120	64	3310
tra10B	50	269	4	670	732	564	20	1186	196	77	6804

3.6 Conclusion

This paper introduces a new problem in the class of capacitated arc routing problems. Two types of vehicles, one for servicing, one replenishing are used. An integer linear formulation is proposed and a cutting plane algorithm is used to solve to optimality

small mixed instances and small to medium size directed instances. For larger problems, the method gives a lower bound (there are still some connectivity constraints that are violated). The size of a problem depends on the SV capacity; if its capacity is small; more replenishments are needed, yielding more variables to model the problem. This article is a first step toward solving more general problems involving several vehicles to be coordinated and various realistic operational constraints. We are currently working on the implementation of heuristics.

Acknowledgment

This work was supported in part by the National Science and Engineering Research Council of Canada and by the *Fonds québécois de la recherche sur la nature et les technologies*.

CHAPITRE 4 : A HEURISTIC METHOD FOR THE CAPACITATED ARC ROUTING PROBLEM WITH REPLENISHMENT: AN APPLICATION TO ROAD MARKING

Article écrit par Alberto Amaya, André Langevin et Martin Trépanier; soumis pour publication à *Journal of the Operational Research Society.*

Ce chapitre présente une extension du problème décrit au chapitre précédant. Dans ce problème, on élimine la contrainte qui oblige au véhicule citerne de retourner au dépôt après qu'il ait rencontré le véhicule traceur de lignes pour faire un ravitaillement. Bref, le véhicule citerne part du dépôt pour visiter plusieurs endroits qui seront des points de ravitaillement. Le véhicule traceur de lignes part aussi du dépôt, il trace des lignes et il trouve sur place le produit de recharge qui a été approvisionné par le véhicule citerne.

Dans cet article, un modèle de programmation mathématique en nombres entiers est d'abord construit, la formulation est inspirée des modèles du CARP, plus précisément du modèle utilisé dans Amaya, et al. (2007). Une formulation du type *sparse* est utilisée, c'est-à-dire le nombre de variables est proportionnel au nombre d'arêtes et arcs dans le graphe. Pour résoudre le problème, le graphe a été augmenté en ajoutant trois ensembles. Le premier ensemble de liens connectent chaque nœud avec le dépôt, le deuxième ensemble de liens connecte le dépôt avec chaque nœud et le troisième ensemble fait un lien entre chaque paire de nœuds (voir section 4.3.1).

Cette formulation a un grand nombre de variables, ce qui fait qu'on est incapable de résoudre en un temps acceptable des problèmes de taille moyenne. Pour pallier ce problème, on a développé une méthode de résolution heuristique. La procédure consiste en trois phases :

- trouver une tournée géante pour le VM ;
- diviser la tournée en sections, où chacune est réalisable pour le VM ;
- sélectionner l'ensemble de sections satisfaisant l'autonomie du VM à un coût minimal.

Pour fabriquer la grande tournée, dans la première phase de l'heuristique, on ignore la contrainte de capacité du véhicule traceur de lignes et on résout un problème du facteur rural (PFR). Lorsque le graphe n'est pas connexe le PFR est NP-dur, mais on peut utiliser une des approches qui ont été proposées dans la littérature. On a utilisé une approche « rendre Connexe-rendre Symétrique », c'est-à-dire, le graphe est tout d'abord rendu connexe et puis symétrique. Cette approche a été utilisée par plusieurs auteurs tels que Christofides (1973).

Une fois que la grande tournée a été faite (appelée W), elle est divisée en sections ou séquences d'arcs, chacune correspondant à un chemin réalisable pour le VM. La partition des tournées prend place sur un graphe orienté noté G*. Le graphe G*= (N*, A*) est défini par son ensemble de nœuds, son ensemble d'arcs, les relations d'incidence et la structure de coût des arcs. Les éléments de G* sont expliqués dans la section 4.4.2.

Une fois le nouveau graphe construit, on calcule le plus court chemin pour aller de la première couche (couche numéro 0) à la dernière couche (couche numéro V). Étant donné que le graphe est un graphe par couches, dans la solution finale on aura un nœud dans chaque couche, ce qui correspond finalement à chaque point de remplissage.

Les tests effectués montrent que l'heuristique peut être utilisée dans un tel contexte. Ce problème ouvre la voie à plusieurs problèmes intéressants de synchronisation de plusieurs véhicules dans le domaine de l'entretien des réseaux routiers et aussi dans divers autres domaines comme le réapprovisionnement d'avions en vol. Des recherches supplémentaires sont nécessaires à la fois dans la modélisation et dans la résolution de tels problèmes.

A Heuristic method for the Capacitated Arc Routing Problem with Replenishment: an Application to Road Marking

Alberto Amaya, André Langevin, Martin Trépanier
GERAD and Department of Mathematics and Industrial Engineering
École Polytechnique de Montréal
C.P. 6079, Succ. Centre-ville, Montréal, Québec, Canada. H3C 3A7

4.1 Abstract

We describe a solution procedure for the Capacitated Arc Routing Problem with Refill Points (CARP-RP). This problem appears from the road network marking in Quebec, Canada. The CARP-RP deals with two different types of vehicles: the first type (called servicing vehicle - SV) which has a finite capacity is used to service the arcs and the other (called refilling vehicle - RV) to replenish the first type of vehicle. The problem consists of simultaneously determining the vehicles routes that minimize the total cost of the two vehicles. In the problem treated here the RV can meet the SV several times before it returns to depot. We present an integer formulation and a route first-second cluster heuristic procedure. Computational results are given.

Keywords: Logistics; Optimization; Networks and graphs; Vehicle routeing; Capacitated arc routeing problem; Heuristics.

4.2 Introduction

The aim of this paper is to present a solution procedure for an extended version of the Capacitated Arc Routing Problem with Refill Points (CARP-RP), introduced by Amaya, et al. (2007). The CARP-RP deals with two different types of vehicles: the first type is

used to service the arcs and the other to replenish the first type of vehicle. The vehicle of the first type (called servicing vehicle - SV), which has a finite capacity, and the vehicle of the second type (called refilling vehicle - RV) can meet at any place in order for the first one to be refilled and to continue its service. The problem consists of simultaneously determining the vehicle route for the SV and the circuit for the RV that minimize the total cost (or distance) while the total demand of any SV path between two consecutives refill points does not exceed its capacity.

The problem stems from road network maintenance where road markings have to be painted or repainted every year, as is the case in the Province of Quebec, Canada. The Quebec Ministry of Transport (MTQ) uses a fleet of special vehicles to mark the roads and also tank trucks to meet the marking vehicle and replenish them.

In a general way, the planning of work for pavement marking on a given network consists in determining how and which segments of roads must be marked (painted), as well as the related schedule. The marking process can be divided in complementary stages. In a first stage, at strategic level, the schedule for road marking improvement is planned. Each season, the road segments and the direction to mark are decided. As an example, in Quebec, the MTQ divides the network into roads (segments and direction of roads) which must be marked each year, roads which must be painted every two years, and roads which must be marked with longer intervals. In other words, at this level the inventory of requirements are defined. For the second stage, at tactical level or seasonal planning, the exact interventions to carry out during one period are defined. At the end of this stage the vehicle circuits are designed. In Quebec the horizon of planning is approximately 6 months (from April to September). And finally for the last stage, at the operational level, the follow-up of operations is done. At this stage, the circuits are dynamically modified in order to react to changes occurred at the last minute and to adjust for the differences between work planed and that carried out.

In the problem presented in Amaya, et al. (2007), the RV has to return to the depot each time it meets the SV. This problem can be seen as an extension of the capacitated arc

routing problem (CARP) introduced by Golden & Wong (1981). The problem, which is NP-hard, was solved using an exact mathematical approach by means of a cutting plane method. However, the method can be used to solve small instances only. In this paper we allow the RV to meet the SV several times before it return to the depot.

The heuristics for the CARP had been broadly classified in Hertz & Mittaz (2000) into three categories: simple constructive methods, two-phase constructive methods (route first-cluster second, and cluster first-route second), and improving methods based on meta-heuristics. Surveys on these algorithms and the various applications of the CARP can be found in Assad & Golden (1995) or Dror (2000). In this paper, after the mathematical formulation, we present a two-phased constructive heuristic to solve the CARP-RP. The heuristic is inspired by the methods proposed in Ulusoy (1985) and Hertz, et al. (2000) for the CARP.

The paper is organized as follows. Section 4.3 presents the mathematical formulation of our version of the CARP-RP. Section 4.4 presents the solution approaches. Computational results are presented in Section 4.5 and conclusions follow in Section 4.6.

4.3 Mathematical formulation

We define the CARP-RP on a directed graph $G=(N; A)$, where $N = \{n_1, \ldots, n_{|N|}\}$ is a set of nodes, including a depot n_{depot} , A is a set of directed arcs $\{a_i \mid a_i = a_{j,k} = (n_j, n_k)$ and $j \neq k\}\}$, including a subset R of required arcs. Each arc of R must be serviced once but they can be traversed several times. The SV has a finite capacity Q. The RV can serve several times the SV before returning to the depot. With each arc a are associated two traversal nonnegative costs:

- c_a: cost incurred by the SV for traversing an arc $a \in A$, $(c_a \geq 0)$
- b_a: cost incurred by the RV for traversing an arc $a \in$ A, $(b_a \geq 0)$

and a nonnegative demand q_a. If $a \notin R$, then $q_a = 0$.

There is also a cost incurred by the SV on servicing required arcs, but this cost does not have any role in solving the CARP-RP as it is a constant because all required arc must be serviced.

The objective is to determine simultaneously the vehicle route ($n_{i1} = n_{depot}, n_{i2}, \ldots, n_{it} = n_{depot}$) for the servicing vehicle (SV) and the vehicle route ($n_{i1} = n_{depot}, \ldots, n_{ij}, \ldots, n_{ik} = n_{depot}$) for the refilling vehicle (RV) that minimize the total costs while the total demand of any SV path between two consecutives refill points does not exceed Q.

To deal with this problem, we have added some arcs transforming the initial graph in a pseudograph as explained in the next subsection.

4.3.1 Pseudograph construction

Construct an auxiliary directed pseudograph $\boldsymbol{H} = (N, L = (A \cup B \cup C \cup D))$, where:

$\boldsymbol{B} = \{ a_{|A|+1}, \ldots, a_{|A|+|N|} / a_i = (n_i, n_{depot}) \}$: set of dummy arcs connecting each node $n_i \in N$ to the depot node. With $c_{ai} = 0$ and $q_{ai} = 0$, $\forall a \in B$:

$\boldsymbol{C} = \{ a_{|A|+|N|+1}, \ldots, a_{|A|+2|N|} \mid a_i = (n_{depot}, n_i) \}$: set of dummy arcs connecting the depot with each node $n_i \in \boldsymbol{N}$. With $c_{ai} = 0$ and $q_{ai} = 0$, $\forall a \in C$.

$\boldsymbol{D} = \{a_{|A|+2|N|+1}, \ldots, a_{|A|+2|N|+|NxN|} \mid a_k = (n_i, n_j) \}$: set of dummy arcs connecting all pair of nodes. And define c_{ak} and q_{ak}, $\forall a_k \in D$, as follows:

$\boldsymbol{c_{ak}}$ = the shortest path cost of going from n_i to n_j, computed with the costs b_a

$q_{ak} = 0$

Figure 4.1 illustrates the original set of arcs A and sets B, C and D added to construct the auxiliary pseudograph H.

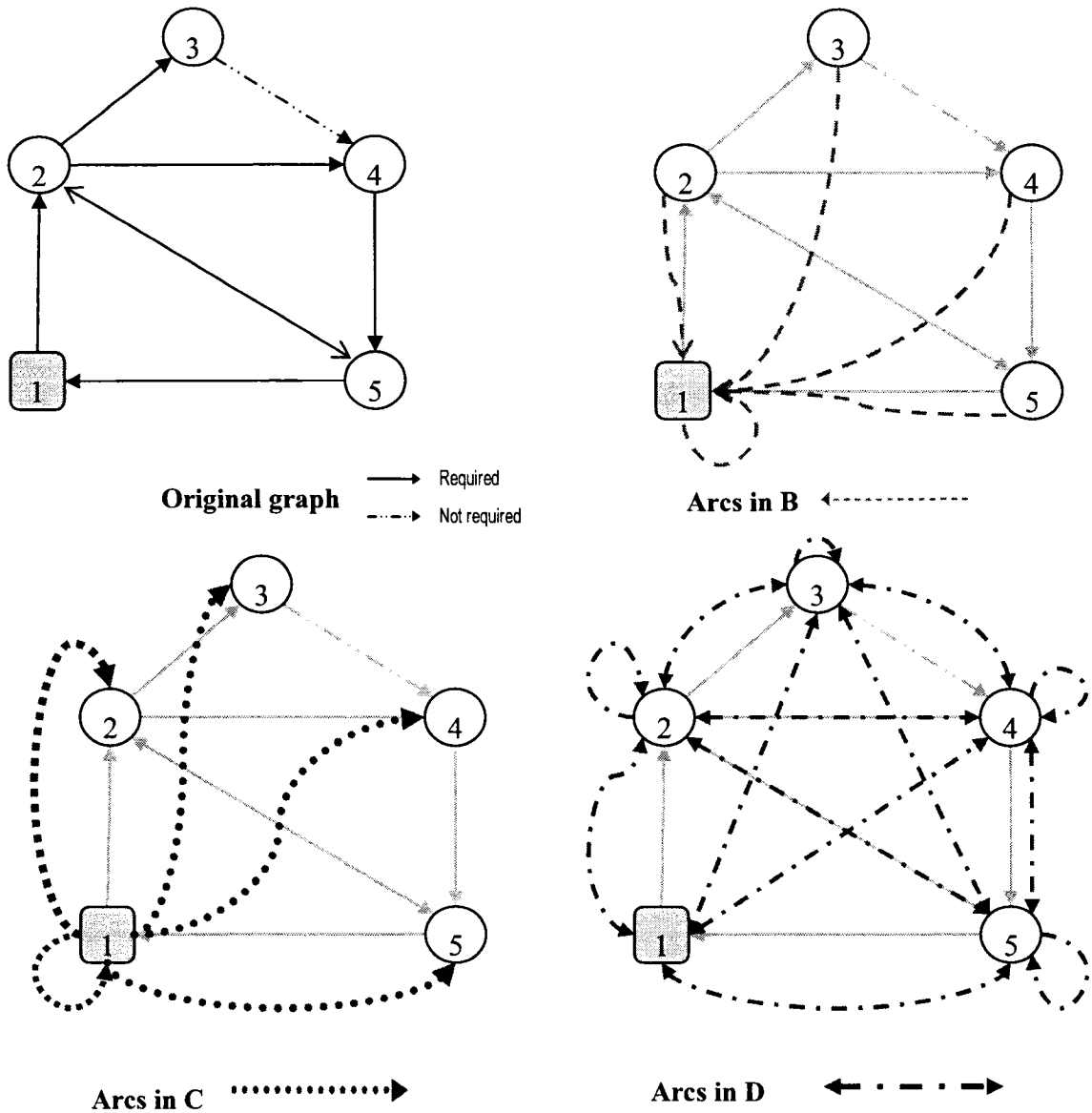


Figure 4.1. Set of arcs of pseudograph H

We also define the following:

- ***O(n)*** $\subseteq$ L: the set of outgoing arcs from node n, $n \in N$
- ***I(n)*** $\subseteq$ L: the set of incoming arcs of node n, $n \in N$
- ***N(S)*** $\subseteq$ N: set of incidents nodes of the arcs of a subset $S \subseteq L$

- ***I(N(S))*** abridged to ***I(S)*** $=\{a / a \in \bigcup_{n \in N(S)} I(n)$ but $a \notin S\}$: the set of arcs entering to $S \subseteq L$
- ***ArtB(n)*** $= B \cap O(n)$: arc of B going out of node n
- ***ArtC(n)*** $= C \cap I(n)$: arc of C coming in node n
- ***ArtD(n,m)*** = index of arc $a \in$ D going from n to m nodes
- ***Q*** : the capacity of SV
- ***P*** : the number of times the SV will be refilled (including the first filling at the depot). This number is assumed to be a positive integer.

And finally let:

W(H) $= \{S \subseteq A$ / $n_{depot} \notin N(S)$, $S \cap R \neq \varnothing$ and the graph $<N(S), S>$ is strongly connected} : the set of strongly connected sub-graphs of H that include at least one required arc but do not include the depot.

The depot node is designed as n_{depot} or simply *depot*.

4.3.2 Decisional variables

We assume that the SV starts at the depot with full capacity and this is the first refill.

Let

$$x_{pa} = \begin{cases} \text{if } a \in \text{A}: & \text{the number of times arc } a \text{ is traversed by the SV, after replenishment } p \text{ and before } p+1. \\ \text{if } a \in \text{B}: & 1 \text{ if the tail of arc } a \text{ is the last node visited by the SV after replenishment } p \text{ and before } p+1;\ 0 \text{ otherwise} \\ \text{if } a \in \text{C}: & 1 \text{ if the head of arc } a \text{ is the first node visited by the SV after replenishment } p \text{ and before } p+1;\ 0 \text{ otherwise} \\ \text{if } a \in \text{D}: & 1 \text{ if arc } a \text{ is used by RV to make replenishment } p+1;\ 0 \text{ otherwise} \end{cases}$$

y_{pa} = 1 if arc $a \in$ R is serviced by the SV after refill p and before refill p+1; 0 otherwise.

Variables y need to be defined only for the required arcs. The total number of variables is then P*(|A|+2*|N|+|N|^2 + |R|).

4.3.3 Formulation

Minimize:

$$\sum_{p=1..P} \sum_{a \in L} c_a x_{pa} \tag{4.1}$$

Subject to:

$\forall p = 1...P, \forall n \in N$: $$\sum_{a \in I(n), a \notin D} x_{pa} - \sum_{a \in O(n), a \notin D} x_{pa} = 0 \tag{4.2}$$

$\forall a \in R$: $$\sum_{p=1..P} y_{pa} = 1 \tag{4.3}$$

$\forall p = 1...P, \forall a \in R$: $$x_{pa} >= y_{pa} \tag{4.4}$$

$\forall p = 1...P$: $$\sum_{a \in B} x_{pa} = 1 \tag{4.5}$$

$\forall p = 1...P$: $$\sum_{a \in C} x_{pa} = 1 \tag{4.6}$$

$\forall p = 1...P$: $$\sum_{a \in R} q_a y_{pa} \leq Q \tag{4.7}$$

$\forall p = 2...P, \forall n \in N$: $$x_{p,ArtC(n)} = x_{p-1,ArtB(n)} \tag{4.8}$$

$$x_{1,ArtC(depot)} = x_{P,ArtB(depot)} = 1 \tag{4.9}$$

$\forall S \in W(H)$, $\forall b \in S \cap R; p = 1...P$: $$\sum_{a \in I(S)} x_{pa} \geq y_{pb} \tag{4.10}$$

$\forall p = 1...P, \forall a \in D$: $$X_{p,ArtC(tail(a))} >= X_{pa} \tag{4.11}$$

$\forall p = 1...P, \forall a \in D$: $$X_{p,ArtB(head(a))} >= X_{pa} \tag{4.12}$$

$$\forall \mathbf{p} = \mathbf{1}...\mathbf{P}, \forall \boldsymbol{a} \in \mathbf{D}: \quad X_{p,ArtC(tail(a))} + X_{p,ArtB(head(a))} <= X_{pa} + 1 \tag{4.13}$$

$$\forall p = 1...P: \quad y_{pa} \in \{0,1\} \forall a \in R \ ; \ x_{pa} \in Z \ \forall a \in A \ ; \ x_{pa} \in \{0,1\} \forall a \in B \cup C \cup D \tag{4.14}$$

The objective function (4.1) represents the total cost incurred by the SV and RV for traversing arcs. The total distance covered by the RV is obtained by summing the c_a ($a \in$D) of the arcs used in the optimal solution. Constraints 4.2 to 4.10 are similar to model defined in Amaya, et al. (2007), that is: the first set of constraints (4.2) is the classic flow conservation constraints. The set of constraints (4.4) ensure that an arc serviced is necessarily traversed. The set of constraints (4.5) and (4.6) requires that only one artificial arc must be traversed on each refill. The set (4.7) defines the SV capacity constraints. The set of constraints (4.8) requires that tour k begins where tour (k-1) finishes. The set of constraint (4.9) forces to begin and finish the SV tour at the depot. Constraints (4.10) are the connectivity constraints. Constraints (4.11), (4.12), and (4.13) are a transformation into linear constraints of the non-linear constraint:

$$\forall p = 1...P, \forall a \in \mathbf{D}: \quad x_{p,ArtC(tail(a))} * x_{p,ArtB(head(a))} = x_{pa} \tag{4.15}$$

These constraints assure that the arc $a \in$D will be active or used (x_{pa} =1) if and only if both the arc belonging to set C that arrives to the tail node of a and the arc belonging to set B who leaves from the head node of a are actives (i.e., $x_{p,ArtC(tail(a))}$ =1 , $x_{p,ArtB(head(a))}$ =1). That set of constraints forces a closed tour for the RV. Constraints (4.14) define the variables.

The integer linear model proposed has an exponential number of constraints. Only small instances can be solved directly in reasonable time. So, we develop a heuristic approach.

4.4 A heuristic algorithm for the CARP-RP

The solution procedure presented herein can be classified as a route first – cluster second approach. The procedure consists of three phases:

- Find a tour for a SV

- Divide the tour into sections, each feasible for the SV
- Select the least cost set of sections satisfying the SV autonomy

4.4.1 Obtaining a tour for the SV

An initial tour is obtained solving a Rural Postman Problem (RPP) for the SV on the graph G, the capacity constraint was not taken into account. Such a tour can be determined by means of the heuristic proposed by Christofides (1973) for the directed rural postman problem. All the arcs on the tour that are not in R are declared as traversed arcs. The first time an arc in R is visited, it is declared as serviced. The next times, it is declared as traversed arc. At the end of this phase we have a closed walk W:

$W = (n_{i0}, a_{i1}, n_{i1}, a_{i2}, n_{i2} \ldots a_{iS}, n_{iS})$; where $\text{tail}(a_{i1}) = n_{i0} = \text{head}(a_{iS}) = n_{iS} = n_{depot}$

4.4.2 Dividing a tour constructed for the SV

Once the tour is obtained, it is divided into sections or sequence of arcs, each corresponding to a feasible SV walk. The partition of the tour takes place on a transformed (P+1)-partite digraph denoted by G*. The number "P" represents the number of refills. The graph G*=(N*,A*) is defined by its node set, arc set, incidence relationships and the arc cost structure. The elements of G* are explained below.

Let $q(n_i, n_j)$ = total demand from n_i to n_j following a walk W.

$$N^* = \bigcup_{k=0..P} N_k :$$

$N_0 = \{n_{i0}\} = \{depot\}$

$N_p = \{n_{ik} \in W \mid n_{ik} = \text{head}(a_{ik}), a_{ik} \in W, q(n_{ir}, n_{ik}) \leq Q, \forall n_{ir} \in N_{p-1}, k > r\}$

Define N_{pmax} as the first set N_p having n_{ik} (= n_{depot})

And let $N_P := N_{pmax} \setminus \{n_{ik} \mid n_{ik} \neq n_{iS}\}$ it means:

$N_P = \{n_{iS} = depot\}$

As a consequence, the set N* is partitioned into layers:

- layers 1 to *P*-1 consisting of all entry nodes of the respective set N_p and
- layers 0 and P contains only the *depot*

Every pair of nodes $\{(n_a, n_b) : n_a \in N_i , n_b \in N_j , j = i+1, i = 0..P\}$ is connected by an arc if $q(n_a,n_b) \leq Q$. The nodes of such a pair are called adjacent entry nodes.

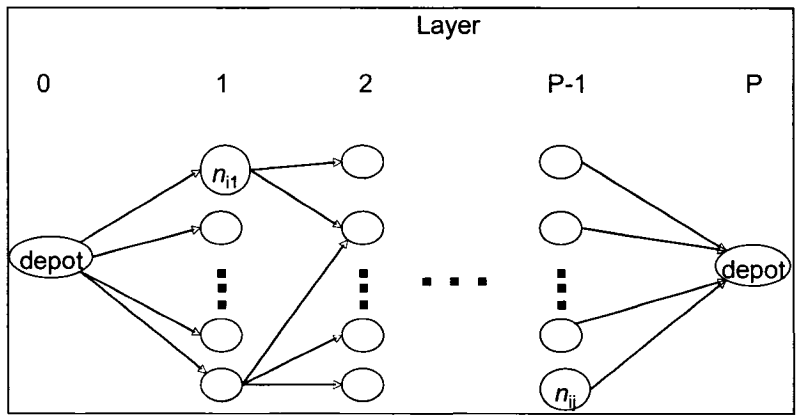


Figure 4.2. Digraph G*

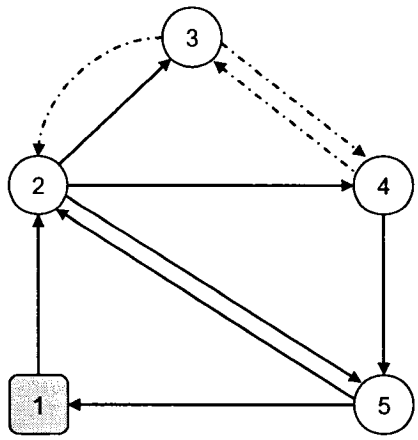
A*: an arc (n_a, n_b) is added between every pair of adjacent entry nodes, the cost is calculated as the sum of $c_{SV}(n_a, n_b)$ and $c_{RV}(n_a, n_b)$:

- $c_{SV}(n_a, n_b)$ = the cost incurred by the SV on the walk W traveling from n_a to n_b
- $c_{RV}(n_a, n_b)$ = the shortest cost on G* incurred by the RV traveling from n_a to n_b. In this problem the RV can leave more than one load at a refilling point, so if n_b is already computed in a predecessor arc, this cost is null.

4.4.2.1 An example

Suppose we have the graph shown in Figure 4.3, the column headings are the arc identification, node head and node tail of arc, and demand and costs as mentioned previously. The continuous lines represent the required arcs (q_a >0). Suppose, after phase 1, we have the following walk: W=(depot,2,3,2,5,2,4,5,depot). Let's take Q=6.

Note that arc (3,2) is not required so its demand is set as null. The service is done the first time an arc is visited. Figure 4.4 presents a graphical representation of this walk, the number near each arc being the demand.



arcs	tale	head	q_a	c_a	b_a
1	1	2	4	20	10
2	2	3	2	20	10
3	2	4	3	20	10
4	2	5	2	30	30
5	3	2	0	20	10
6	3	4	0	20	10
7	4	3	0	20	10
8	4	5	3	20	10
9	5	1	2	20	10
10	5	2	4	30	30

Figure 4.3. Graph and data used in example 1

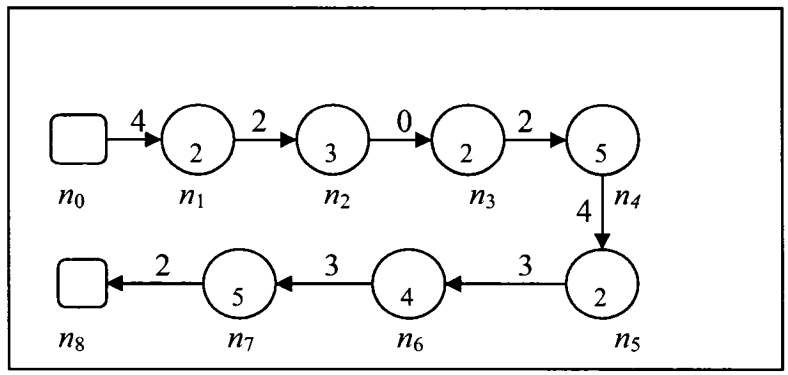


Figure 4.4. W in example 1

From Figure 4.4, the total demand is 20, and the walk cost for the SV is 180 (sum of c_a).

Let index each node in W as follows: W= (n_0=n_{depot}, n_1, n_2,..., n_8 = n_{depot}),

N^* is determinate as follows:

$N_0 = \{ n_0 = depot\} = \{1\}$

$N_1 = \{ n_1, n_2, n_3\} = \{2,3,2\}$

$N_2 = \{ n_2, n_3, n_4, n_5\} = \{3,2,5,2\}$

$N_3 = \{ n_3, \ldots , n_7\} = \{2, \ldots ,5\}$

$N_4 = \{ n_4, \ldots , n_8 = depot\} = \{5, \ldots , 1\}$

Because N_4 is the first set containing the depot, we update $N_4 = \{n_8 = \text{depot}\}$.

Once N* is constructed, we connect every pair of nodes $\{(n_a, n_b)$: $n_a \in N_i$, $n_b \in N_j$, $j = i + 1, i = 0..3\}$ if $q(n_a,n_b) \leq 6$. Figure 4.5 presents the final G* graph.

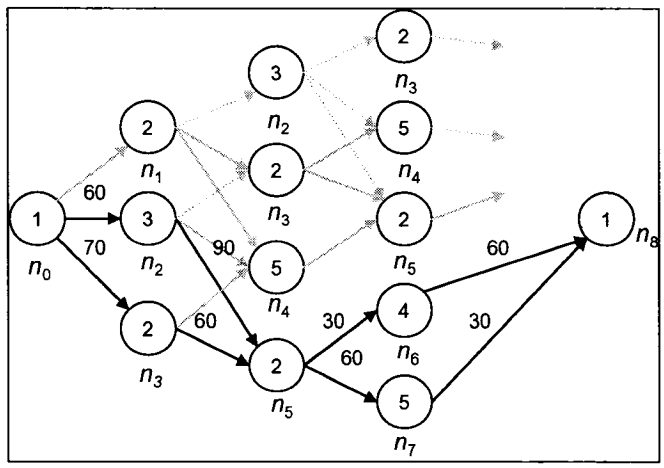


Figure 4.5. Digraph G*

Note that we don't have to compute the cost on the paths not ending at node n_8.

Table 4.1 presents the total cost calculated for Figure 4.5.

Table 4.1. Costs for digraph G*

(n_i, n_j)	Path	$c_{SV}(n_i, n_j)$	Path	$c_{RV}(n_i, n_j)$	Total Cost
(n_0, n_2)	(1,2,3)	40	(1,2,3)	20	60
(n_0, n_3)	(1,2)	60	(1,2)	10	70
(n_2, n_5)	(3,2,5,2)	80	(3,2)	10	90
(n_3,n_5)	(2,5,2)	60	(2,2)	0	60
(n_5, n_6)	(2,4)	20	(2,4)	10	30
(n_5, n_7)	(2,4,5)	40	(2,4,5)	20	60
(n_6, n_{depot})	(4,5,depot)	40	(4,5,1)	20	60
(n_7, n_{depot})	(5,depot)	20	(5,1)	10	30

The first column in the table contains the pairs of adjacent entry nodes. Column 2 and 4 contain the paths followed by the SV and RV as explained before, and columns 3 and 5 contain the respective cost to follow that path. The last column contains the total costs; the total costs are calculated over the relevant arcs in Figure 4.5.

Using the procedure described earlier produces a large number of nodes in each layer (i.e., card(N_v)). This number can be reduced by redefining the set N_v (defined in section 4.4.2) by taking into account that the minimum number of refills can be computed as follows:

$V=\lceil \Sigma q_a/Q \rceil$, so the new set can be calculated as :

$$N_v^* = N_v \cap \{ n_{ik} \mid n_{ik} = \text{head}(a_{ik}),\ q(n_{ip},n_{ik}) \geq q(n_{ik},depot) - Q^*\lfloor q(n_{ik},depot)/Q \rfloor ,\ \forall\, n_{ip} \in N_{v-1} ,\ i > p\}$$

This definition reduces the number of nodes in the set by removing those nodes that if used as refilled points will generate more than $\lfloor q(n_{ik},depot)/Q \rfloor$ refillings points. This definition can give an empty set; for instance, if we have 5 arcs all with demand equal to 4, and Q = 5, we need 5 refills while this formula computes to 4 (a Bin Packing Problem has to be solved in fact). If the set is null we have to use the original set N_v.

4.4.3 Selecting the least cost set of sections

Having constructed G*, we are ready to select the least cost set of sections represented on G*. This is accomplished by solving a shortest path between the node in layer 0 and the node in layer P in graph G*. By construction, the shortest path contains one node in each layer, each of this nodes correspond to one refilling point. The set of arcs between two consecutive refilling points, on W, corresponds to one section. At the end of this phase, we have a feasible tour for the SV and the refilling points to be visited by the RV.

In the example of Figure 4.5, a shortest path is: (n_0, n_3, n5, n_6, n_8) corresponding to the nodes: (depot, 2, 2, 4, depot), so the refilling points are at node number 2 and at node number 4. The total cost is 220, 180 for SV and 40 for the RV.

4.5 Computational experiments

First we tested the integer lineal programming on some problems using the procedure (only phase 2) and instances mentioned in Amaya, et al. (2007). In that paper a set of 180 instances with 3, 4 and 5 replenishments were randomly generated. Due to the resolution time to find an optimal solution, only the problems with 3 replenishments were considered (60 problems). Table 4.2 presents the results. The column headings are as follows:

- N: number of nodes
- A: number of arcs
- time, av. iter: average time in each iteration (in seconds)
- time, av. tot: average total time of procedure (in seconds)
- time, max tot: maximum time in each iteration to find an optimal solution
- iterations, av: average number of iterations to find the optimal solution
- iteration, max: maximum number of iterations to find an optimal solution
- #opts: the number of problems (over 10) where the solution was proved as optimal
- max gap: maximum distance found between upper and lower bounds, in percentage

Table 4.2. Numerical results for the Amaya set of problems (exact approach)

		Time (sec)			iterations			
N	A / N	av. iter	av. tot	max tot	av	max	# opts	max gap
[20,30]	[2.5 , 4.5]	10.51	21.79	20.94	2.0	6	9	1.2%
	[6.5 , 8.5]	31.97	33.33	59.96	1.1	2	10	0.0%
[40,50]	[2.5 , 4.5]	34.12	74.62	66.02	2.3	5	9	0.9%
	[6.5 , 8.5]	46.66	46.66	93.81	1.0	1	10	0.0%
[60,70]	[2.5 , 4.5]	134.65	327.84	264.28	2.8	7	9	0.8%
	[6.5 , 8.5]	139.65	139.65	252.58	1.0	1	10	0.0%

Using the procedure we were able to solve at least 57 of the 60 problems. The optimal value found was used to test the heuristic.

The results obtained from applying the heuristic to the set of problems solved to optimality are shown in Table 4.3. The table presents for each group of problems the average time employed by the heuristic (av. time), the average gap found, in percentage, between the optimal value and the value found by the heuristic (av. gap), and the maximum gap found in each group of problems (max gap).

Table 4.3. Numerical results using the heuristic

N	A / N	av. time	av. gap	max gap
[20,30]	[2.5 , 4.5]	17.7	3.7%	8.5%
	[6.5 , 8.5]	51.3	0.6%	1.9%
[40,50]	[2.5 , 4.5]	20.3	2.8%	5.1%
	[6.5 , 8.5]	96.1	1.5%	3.5%
[60,70]	[2.5 , 4.5]	137.5	0.9%	2.7%
	[6.5 , 8.5]	381.5	0.5%	1.6%

This procedure led to long computation time calculating the relevant cost between each pair of adjacent nodes. We used a new strategy to accelerate the solution of the CARP-rp. The strategy works as follow.

1. Set k = V, so $N_k = \{ n_{iS} = depot\}$ the last set (i.e., layer) found in section 4.4.2 and take the only node in layer k (the depot) as n_{target}
2. Determine the cost of all arcs (u, n_{target}) such that $u \in N_{k-1}$. Add some of them to a graph G1, for example those with the smallest costs.
3. Set $n_{target} = u$, those node sources found in step 2.
4. If $k = 2$, stop. Otherwise, set $k = k - 1$ and return to 2.

Figure 4.6. Strategy of acceleration

The previous strategy improves the resolution time, but it makes worse the total cost. In order to test the gap between the total costs obtained, we tested the heuristics with all the Amaya set of problems and different percentages of arcs added in step 2.

The first strategy consists in adding only one arc in step 2, the one with the lowest cost. Table 4.4 presents that case, this table presents for each group of problems the average

time employed by the improved heuristic (av. time), the average gap, in percentage, between the total cost for the heuristic and the improved heuristic (av. gap), and the maximum gap found in each group of problems (max gap).

Table 4.4. Numerical results adding only one arc

N	A / N	RF	av. time	av gap	max gap
[20,30]	[2.5 , 4.5]	3	0.39	2.9%	12.2%
		4	0.20	5.0%	11.0%
		5	0.28	7.8%	15.0%
	[6.5 , 8.5]	3	0.37	2.1%	4.7%
		4	0.42	2.8%	5.4%
		5	0.38	2.6%	4.6%
[40,50]	[2.5 , 4.5]	3	0.23	0.5%	3.5%
		4	0.39	4.6%	6.5%
		5	0.35	3.5%	7.2%
	[6.5 , 8.5]	3	0.79	2.0%	4.6%
		4	0.83	2.1%	4.3%
		5	0.64	2.2%	4.2%
[60,70]	[2.5 , 4.5]	3	0.77	2.0%	3.5%
		4	0.51	1.6%	3.6%
		5	0.63	3.2%	5.5%
	[6.5 , 8.5]	3	1.01	0.5%	1.9%
		4	1.54	1.2%	2.2%
		5	1.50	1.6%	2.4%

Three more strategies of acceleration were tested, the percentage of arcs added were: 25%, 50% and 75%. Figure 4.7 presents the results for these cases, also the case where only one arc is added and the case without strategy of acceleration. The results are grouped in three sets: set A is the first 60 problems in Table 4.4, set B is the next 60 and C the last 60 problems. The figure presents the gap between the heuristic without acceleration and each strategy and the resolution time in each case (average for the 60 problems).

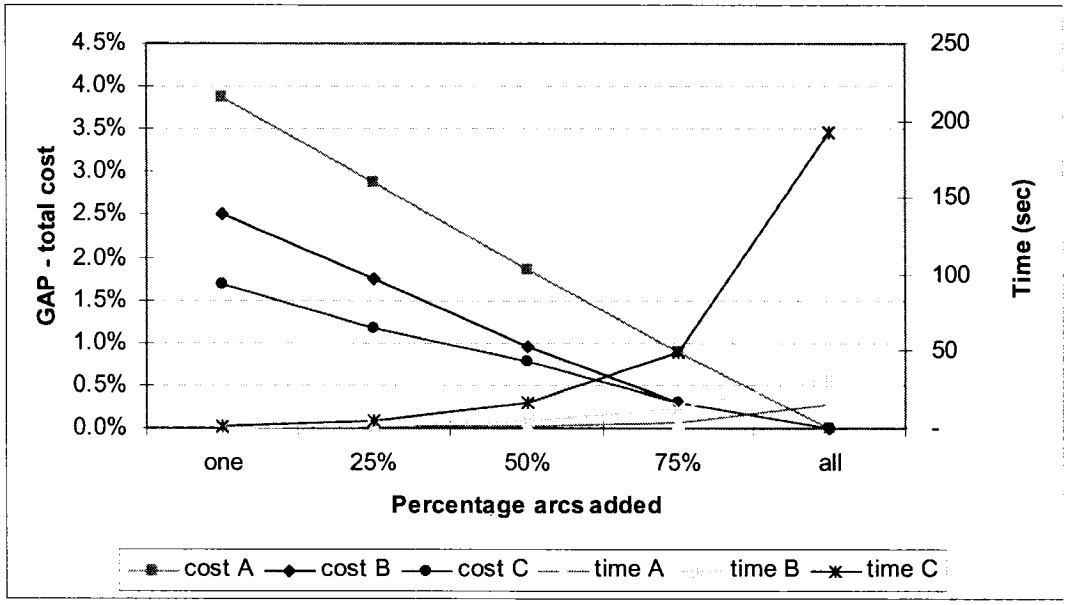


Figure 4.7. Results strategy of acceleration

This figure shows that using small percentages of arcs in the strategy of acceleration can improve the resolution time, losing less than 4.0% of total cost for the cases with graphs with 20 to 30 nodes and less than 2% for the cases with 60 to 70 nodes.

4.6 Conclusions

In this paper, a special CARP is presented. The problem is inspired by the road network maintenance where the road markings have to be painted or repainted every year. An integer linear formulation is proposed and a cutting plane algorithm is used to solve to optimality small size directed instances. In addition, a two-phase constructive method is presented and a strategy of acceleration as well. The heuristic seems to work well in the problems tested. Several extensions are still needed to make the model more useful in practice. For example, multi-RV and multi-PV can be implemented. Other extensions and post optimization methods can be apply to the heuristic.

CHAPITRE 5 : LE PROBLÈME DE TOURNÉES SUR LES ARCS AVEC POINTS DE REMPLISSAGE ET AVEC PLUSIEURS VÉHICULES DE MARQUAGE ET DE RÉAPPROVISIONNEMENT.

5.1 Résumé

Dans ce chapitre, nous présentons le problème de tournées sur les arcs avec points de remplissage avec plusieurs véhicules de réapprovisionnement et plusieurs véhicules de marquage. Nous décrivons un modèle de programmation linéaire en nombres entiers. Ce modèle généralise les deux problèmes traités dans les deux chapitres précédents. Dans ce problème, deux types de véhicules sont utilisés. Les véhicules du premier type (véhicules de marquage) servent à tracer les lignes sur la chaussée et les véhicules du deuxième type (véhicules de réapprovisionnement) servent à réapprovisionner les véhicules de marquage. Le problème consiste à déterminer simultanément les routes des véhicules de marquage et celles des véhicules de réapprovisionnement. Une méthode de résolution exacte et une procédure heuristique sont décrites.

5.2 Introduction

Rappelons brièvement les deux problèmes précédents. Dans le chapitre 3, une formulation en nombres entiers et une méthode de résolution exacte furent données pour résoudre le problème d'un seul véhicule de marquage (VM) et un seul véhicule de réapprovisionnement (VR). Dans ce problème, le véhicule de réapprovisionnement doit revenir au dépôt chaque fois qu'il a fait un remplissage. La formulation mathématique est basée sur les formulations du CARP. Les arêtes sont remplacées par deux arcs. Deux types de variables sont utilisés : X_{va}, qui représente le nombre de fois où le VM est passé par l'arc a après le réapprovisionnement p et avant le $p+1$, (c.-à-d. le trajet p) et Y_{va}, qui

égale 1 si l'arc a est desservi dans le trajet p. La méthode de résolution est une méthode classique de plans de coupe similaire à celle de Lacomme, et al. (2002). La méthode consiste à résoudre un problème de programmation en nombres entiers P' qui omet toutes les contraintes de connectivité du problème original P (il y a un nombre exponentiel de ces contraintes). Si la solution de P' satisfait les contraintes de connectivité, nous avons une solution optimale pour le CARP-RP, sinon les contraintes de connectivité violées sont ajoutées à P' et il est résolu de nouveau. Cependant, compte tenu de la complexité du modèle, seules de petites instances peuvent être résolues en un temps raisonnable.

Dans le chapitre 4, une formulation en nombres entiers et une méthode de résolution heuristique sont données pour résoudre le problème d'un seul véhicule de marquage et un seul véhicule de réapprovisionnement. Dans ce problème, le véhicule de réapprovisionnement peut réapprovisionner le véhicule de marquage plusieurs fois avant de revenir au dépôt. Le problème consiste à déterminer simultanément la route du véhicule de marquage et celle du véhicule de réapprovisionnement. La procédure heuristique développée consiste en trois étapes : trouver une tournée géante pour le VM, diviser la tournée en sections, où chacune est réalisable pour le VM, et sélectionner l'ensemble de sections satisfaisant l'autonomie du VM à un coût minimal. Une variation de la méthode est présentée, ce qui permet de trouver une solution plus rapidement sans compromettre de plus de 4% le coût total (dans les problèmes testés).

5.3 Formulation Mathématique

5.3.1 Définition du problème

Le problème est une généralisation du CARP-RP où plusieurs véhicules de marquage et plusieurs véhicules de remplissage sont utilisés. Nous l'appelons le problème de tournées sur les arcs avec points de remplissage et plusieurs véhicules. Dans cette situation, nous disposons de V véhicules de marquage (VM) et de K véhicules de remplissage (VR). Le réseau est desservi utilisant les VM, chaque VM ayant une

capacité finie et pouvant être rempli sur place par un véhicule de remplissage (VR). Chaque VM peut être réapprovisionné par n'importe quel VR et peut être rempli plusieurs fois avant de revenir au dépôt. Les VR peuvent remplir les VM plusieurs fois avant de revenir au dépôt.

Considérons un graphe orienté $G= (N; A)$, où $N = \{n_1, \ldots, n_{|N|}\}$ est un ensemble de nœuds, incluant le dépôt $n_{dépôt}$, A un ensemble d'arcs orientés $\{a_i \mid a_i = a_{j,k} = (n_j, n_k)$ et $j \neq k\}\}$, dont un sous-ensemble R d'arcs obligatoires. Chaque arc de R doit être desservi une fois mais peut être emprunté plusieurs fois. Chaque VM a une capacité maximale Q. Chaque VR peut remplir les VM au maximum F fois (un nombre fini de fois) avant de revenir au dépôt. À tout arc a sont associés une demande non négative q_a (si $a \notin R$, alors $q_a = 0$) et deux coûts non négatifs de passage :

- c_a : le coût de passage du VM sur l'arc $a \in A$, $(c_a \geq 0)$;
- b_a : le coût de passage du VR sur l'arc $a \in$ A, $(b_a \geq 0)$.

L'objectif est de déterminer simultanément les tournées des VM et VR sur le graphe G minimisant le coût total, tel que pour chaque VM :

- la tournée commence et finit au dépôt ;
- le nombre maximal de réapprovisionnements est P ;
- la demande servie entre deux réapprovisionnements n'excède pas sa capacité (Q).

Et pour chaque VR :

- la tournée commence et finit au dépôt ;
- le nombre de fois qu'il approvisionne n'excède pas sa capacité (F).

Pour traiter ce problème, le graphe G est transformé en un graphe augmenté, comme proposé dans le chapitre 4. La section suivante explique la construction du graphe augmenté.

5.3.2 Construction d'un graphe augmenté

Le graphe original est modifié en y ajoutant des arcs artificiels. Soit le graphe orienté $\boldsymbol{H}$ = (N, L= ($A \cup B \cup C \cup D$)), où :

- $\boldsymbol{B}$ = { $a_{|A|+1}$, … ,$a_{|A|+|N|}$ / a_i = (n_i , $n_{dépôt}$) } : ensemble d'arcs artificiels liant chaque noeud $n \in N$ au nœud dépôt, avec c_{ai} =0 et q_{ai} =0, $\forall a \in B$;

- $\boldsymbol{C}$ = { $a_{|A|+|N|+1}$, … ,$a_{|A|+|N|+|N|}$ | a_i = (n_{depot} , n_i) } : ensemble d'arcs artificiels liant le dépôt à chaque noeud $n \in N$, avec c_{ai} = 0 et q_{ai} = 0, $\forall a \in C$;

- $\boldsymbol{D}$ = {$a_{|A|+2|N|+1}$, … , $a_{|A|+2|N|+|NxN|}$ | a_k = (n_i , n_j) } : ensemble d'arcs artificiels liant chaque paire de nœuds. Et définissons b_{ak} et q_{ak} , $\forall a_k \in D$, comme suit :

  b_{ak} : le coût du plus court chemin de n_i à n_j, calculé avec les coûts b_a.;

  q_{ak} = 0.

La Figure 4.1 illustre le graphe original et les ensembles B, C et D du graphe H. De plus, soit :

- ***O(n)*** $\subseteq L$: l'ensemble des arcs issus du noeud $n \in N$;
- ***I(n)*** $\subseteq L$: l'ensemble des arcs se terminant au noeud $n \in N$;
- ***N(S)*** $\subseteq N$: l'ensemble des nœuds incidents aux arcs d'un sous-ensemble $S \subseteq L$;
- ***I(N(S))*** $\equiv$ ***I(S)*** $= \{a / a \in \bigcup_{n \in N(S)} I(n)$ et $a \notin S\}$: l'ensemble des arcs entrants dans l'ensemble $S \subseteq L$;
- ***ArtB(n)*** = $B \cap O(n)$: arc de B issu du noeud n ;

- ***ArtC(n)*** = $C \cap I(n)$: arc de C se terminant au noeud n ;
- ***ArtD(n,m)*** = index de l'arc $a \in D$ du nœud n au nœud m ;
- ***W(H)*** = $\{S \subseteq A$ / $n_{dépôt} \notin N(S)$, $S \cap R \neq \varnothing$, le graphe $<N(S), S>$ est fortement connexe$\}$: l'ensemble de sous-graphes de H fortement connexes qui incluent au moins un arc obligatoire mais n'incluent pas le dépôt ;
- $V_M = \{ v_1, ..., v_V\}$: l'ensemble de véhicules de marquage ;
- $V_R = \{ v_{V+1}, ..., v_{V+K}\}$: l'ensemble de véhicules de remplissage.

Le nœud dépôt est noté $n_{\text{dépôt}}$ ou simplement *dépôt.*

5.3.3 Variables de décision

Assumons que le VM commence au dépôt à pleine capacité et que ceci est le premier réapprovisionnement.

Soit :

$$x^{v}_{pa} = \begin{cases} \text{Si } a \in \text{A : Nombre de fois que l'arc } a \text{ est traversé par le } v\text{-ième VM, après son remplissage } p \text{ et avant le remplissage } p+1 \text{ (trajet } p\text{) où } p = 1...\text{P}, v \in V_{\text{VM}} ; \\ \text{Si } a \in \text{B : 1 si la fin de l'arc } a \text{ est le dernier noeud visité par le } v\text{-ième VM dans le trajet } p \text{ ; 0 sinon où } p = 1...\text{P}, v \in V_{\text{VM}} ; \\ \text{Si } a \in \text{C : 1 si le début de l'arc } a \text{ est le premier noeud visité par le } v\text{-ième VM dans le trajet } p \text{ ; 0 sinon où } p = 1...\text{P}, v \in V_{\text{VM}} ; \\ \text{Si } a \in \text{D : 1 si le } k\text{-ième VR passe par l'arc } a \text{ pour faire le réapprovisionnement } v+1 \text{ ; 0 sinon où } p = 1...\text{P}, v \in V_{\text{VR}}. \end{cases}$$

y^{v}_{pa} = 1 si l'arc $a \in$ R est desservi par le v-ième VM dans le trajet p; 0 sinon.

Les variables *y* doivent être définies seulement pour les arcs obligatoires.

5.3.4 Formulation

Minimiser :

$$\sum_{v\in V_M}\sum_{p=1}^{P}\sum_{a\in L\setminus D} c_a x_{pa}^v + \sum_{v\in V_R}\sum_{p=1}^{P}\sum_{a\in D} b_a x_{pa}^v \qquad \textbf{(5.1)}$$

Sous les contraintes :

$\forall v \in V_M, \forall p = 1...P, \forall n \in N$: $$\sum_{a\in I(n), a\notin D} x_{pa}^v - \sum_{a\in O(n), a\notin D} x_{pa}^v = 0 \qquad \textbf{(5.2)}$$

$\forall a \in R$: $$\sum_{v\in V_M}\sum_{p=1}^{P} y_{pa}^v = 1 \qquad \textbf{(5.3)}$$

$\forall v \in V_M, \forall p = 1...P, \forall a \in R$: $$x_{pa}^v >= y_{pa}^v \qquad \textbf{(5.4)}$$

$\forall v \in V_M, \forall p = 1...P$: $$\sum_{a\in B} x_{pa}^v = 1 \qquad \textbf{(5.5)}$$

$\forall v \in V_M, \forall p = 1...P$: $$\sum_{a\in C} x_{pa}^v = 1 \qquad \textbf{(5.6)}$$

$\forall v \in V_M, \forall p = 1...P$: $$\sum_{a\in R} q_a y_{pa}^v \leq Q \qquad \textbf{(5.7)}$$

$\forall v \in V_M, \forall p = 2...P, \forall n \in N$: $$x_{p,ArtC(n)}^v = x_{p-1,ArtB(n)}^v \qquad \textbf{(5.8)}$$

$$x_{1,ArtC(depot)}^v = x_{P,ArtB(depot)}^v = 1 \qquad \textbf{(5.9)}$$

$\forall W \in W(H), \forall b \in W \cap R, \forall v \in V_M, \forall p = 1...P$:

$$\sum_{a\in I(W)} x_{pa}^v \geq y_{pb}^v \qquad \textbf{(5.10)}$$

$\forall v \in V_R$: $$\sum_{a\in D}\sum_{p=1}^{k} x_{pa}^v <= F \qquad \textbf{(5.11)}$$

$\forall v \in V_M, \forall p=1...P, \forall a \in D:$ $$x^v_{p,ArtC(tail(a))} >= \sum_{v \in V_R} x^v_{pa} \quad (5.12)$$

$\forall v \in V_M, \forall p=1...P, \forall a \in D:$ $$x^v_{p,ArtB(head(a))} >= \sum_{v \in V_R} x^v_{pa} \quad (5.13)$$

$\forall v \in V_M, \forall p=1...P, \forall a \in D:$ $$x^v_{p,ArtC(tail(a))} + x^v_{p,ArtB(head(a))} \leq \sum_{v \in V_R} x^v_{pa} + 1 \quad (5.14)$$

$\forall v \in V_M, \forall p = 1...P:$ $$y_{va} \in \{0,1\} \forall a \in R \quad (5.15)$$

$\forall v \in V_M, \forall p = 1...P:$ $$x^v_{pa} \in Z \; \forall a \in A \; ;$$

$$x^v_{pa} \in \{0,1\} \forall a \in B \cup C \quad (5.16)$$

$\forall v \in V_R, \forall p = 1...P:$ $$x^v_{pa} \in \{0,1\} \forall a \in D \quad (5.17)$$

Ce problème comporte trois coûts importants : le coût de desservir les arcs obligatoires, le coût de passage du VM sur les arcs et le coût de passage du VR sur les arcs du graphe. Le premier coût n'a aucune influence sur la résolution du problème. Ce coût est une constante puisque tous les arcs obligatoires doivent être desservis (contraintes 5.3). La fonction-objectif (5.1) représente le coût total engendré par le passage des VM et des VR sur les arcs. La distance totale parcourue par les VM est obtenue en faisant la somme du premier terme de la fonction-objectif et la distance totale parcourue par les VM est obtenue en faisant la somme du deuxième terme de la fonction-objectif dans la solution optimale. Les contraintes (5.2) sont des contraintes classiques de conservation de flux à chaque nœud d'un réseau. L'ensemble de contraintes (5.4) assure qu'un arc desservi est nécessairement traversé. Les contraintes (5.5) et (5.6) indiquent qu'un seul arc artificiel doit être traversé pour chaque réapprovisionnement. L'ensemble (5.7) définit les contraintes d'autonomie du VM. L'ensemble (5.8) précise que le trajet *p* du véhicule *v* commence là où le trajet *p-1* de ce même véhicule se termine. L'ensemble de contraintes (5.9) force les VM à commencer et terminer leur tournée au dépôt. Les contraintes (5.10)

sont les contraintes de connectivité. La contrainte (5.11) délimite le nombre de fois que les VR peuvent remplir les VM dans leur tournée. Les contraintes (5.12, 5.13 et 5.14) sont une transformation en contraintes linéaires de la contrainte non linéaire suivante :

$$\forall v \in V_M, \forall p=1...p, \forall a \in D: \; x^{v}_{p,ArtC(tail(a))} * x^{v}_{p,ArtB(head(a))} = \sum_{v \in V_R} x^{v}_{pa} \qquad (5.18)$$

Cette famille de contraintes force les VR à faire des tournées fermées. Les contraintes (5.14) à (5.17) définissent les variables.

Dans le modèle précédent le nombre de véhicules de marquage et le nombre de véhicules de remplissage sont connus au départ. Dans des applications réelles de marquages de réseaux routiers le nombre de véhicules est inférieur au nombre de tournées totales à faire pour accomplir le service. Dans ce cas, le nombre de véhicules du modèle correspond au nombre total de tournées réalisées. Ce qui est une variable inconnue pour le modèle. Cette situation peut être rectifiée en calculant à priori le nombre de véhicules et en ajoutant au modèle des contraintes de pénalisation d'utilisation des véhicules. Pour trouver une borne inférieure sur le nombre de véhicules, on peut utiliser la borne pour le problème de sac à dos (*Bin Packing Problem*). Cette borne est au moins aussi bonne que $\lceil \Sigma q_{ij} / Q \rceil$ mais elle nécessite des temps de calcul assez importants sur des instances de grande taille.

5.4 Méthodes de résolution

Pour résoudre le modèle (1-14), une méthode classique de plans de coupe similaire à celle utilisée au chapitre 3 pourrait être utilisée. Cette méthode est facile à mettre en œuvre. Une méthode de *Branch and Cut* peut aussi être utilisée mais avec un temps de développement nettement supérieur. Cependant, compte tenu de la complexité du modèle, seules de petites instances peuvent être résolues en un temps raisonnable. Aussi, pour les problèmes de grande taille, une approche heuristique doit être utilisée.

5.4.1 Approche heuristique de résolution

Un cadre de résolution est présenté dans ce qui suit. La procédure de solution est une méthode heuristique du type *cluster first – route second* codée dans une méthode à plusieurs phases inspirée des méthodes proposées par Ulusoy (1985), Hertz (2005) et Benavent, et al. (1990) pour le CARP et la méthode proposée au chapitre précédent pour le PTAC-PR. Pour une étude plus approfondie sur le CARP, on peut consulter Assad & Golden (1995) ou Dror (2000). Les phases de l'approche sont les suivantes.

Heuristique : Construction - CARP_RP

Phase 1. Partitionnement des arcs requis

Dans cette étape, nous déterminons S secteurs, avec approximativement la même charge de travail, qui correspond à une partition du réseau en tenant compte des arcs requis. Chaque secteur est affecté à un VM. Cette procédure peut être réalisée en utilisant une des techniques de partitionnement de graphes telles que celle proposée par Benavent, et al. (1990) pour le CARP. Dans cette approche, un problème d'affectation généralisé (*generalized assignment problem* (GAP)) est résolu; la technique est inspirée de Ficher & Jaikumar (1981) pour le problème de tournées de véhicules avec capacité (*capacitated vehicle routing problem*).

Phase 2. Détermination de points de remplissage et des tournées pour chaque VM

À cette étape, pour chaque secteur nous déterminons des trajets et les points de remplissage. Cette procédure peut se réaliser en utilisant la méthode proposée dans cette thèse pour le problème d'un seul VM et un seul VR. Dans cette approche, nous trouvons d'abord une tournée géante (PPR) pour le VM en utilisant une technique comme celle proposée par Christofides, et al. (1986), puis nous divisons la tournée en trajets réalisables pour le VM en utilisant une des techniques développées au chapitre précédent.

Phase 3. Déterminer les tournées pour les VR

Avec tous les points de remplissages trouvés à la phase 2, nous résolvons un problème de tournées de véhicules avec capacité. Chaque point de remplissage correspond à un nœud requis, avec une demande égale au nombre de fois que ce nœud est un point de remplissage. Nous utilisons une flotte de K véhicules, qui correspondent aux VR. Cette phase peut être réalisée en utilisant une approche exacte, parce que le nombre de points de remplissage n'est pas très élevé. Ce problème a été étudié dans les 40 dernières années et de multiples méthodes exactes et heuristiques ont été développées. Voir par exemple Toth & Vigo (2001) ou Cordeau, et al. (2005).

Le cadre présenté ci-dessus ne correspond pas à une méthode fixe, cette méthode de résolution peut varier et les phases peuvent être jumelées. Par exemple, les phases 1 et 2 peuvent être assemblées en une seule en utilisant une méthode de résolution pour le CARP. Si une telle méthode est utilisée, la solution du CARP donne l'affectation des arcs demandés dans la phase 1 et la tournée demandée au début de la phase 2. Cette stratégie a été utilisée dans le cas d'étude présenté ci-après.

5.5 Expérimentation

La procédure heuristique décrite a été utilisée sur le réseau de Marzolf (2003). Ce réseau est un graphe totalement orienté et représente une partie de la région administrative de l'Estrie. Le réseau possède les caractéristiques suivantes :

Tableau 5.1. Caractéristiques du réseau utilisé

	Nombre	Longueur (Km)	Moyenne (Km)	Longueur Min (Km)	Longueur Max (Km)
Noeuds	140				
Liens	374	2944,9	7,9	2,3	40,8

Les paramètres utilisés pour les calculs ne correspondent pas exactement aux données réelles, mais ils correspondent approximativement aux attributs de l'opération saisonnière :

- le nombre de kilomètres à parcourir par véhicule dans une tournée ne doit pas excéder 350 ;
- les véhicules marqueurs doivent être remplis au plus 4 fois ;
- les véhicules de réapprovisionnement peuvent servir jusqu'à 4 fois des véhicules marqueurs avant de revenir au dépôt ;
- tous les liens du réseau doivent être marqués (requis).

Phase 1 :

Dans la phase 1 de l'heuristique, le nombre de véhicules de marquage doit être connu. Dans le cas étudié, nous devons trouver ce nombre à partir du nombre de tournées à réaliser. Pour déterminer le nombre minimal de véhicules de marquage nécessaires (donc, de tournées), un problème de type « sac à dos » doit être résolu. Dans la pratique, il s'agit d'un problème très difficile, qui fait partie des problèmes NP-complets.

Pour déterminer le nombre de tournées, nous avons utilisé une méthode basée sur la méthode *Augmented-Insert* de Pearn (1991) qui résout le problème du CARP. L'approche est inspirée de celle développée par Clarke & Wright (1964) pour le problème de tournées sur les sommets avec capacité : partant d'un ensemble où chaque tournée ne dessert qu'un seul arc, le nombre de véhicules utilisés est progressivement réduit. L'objectif utilisé est la minimisation de la distance parcourue par tous les véhicules de marquage. Cette méthode donne le nombre de véhicules de marquage nécessaires, l'affectation des arcs aux tournées et une tournée « géante » nécessaire au début de la phase 2.

Le graphique suivant présente l'affectation trouvée à la phase 1. Neuf secteurs furent trouvés et chacun est représenté par une teinte distincte dans la figure.

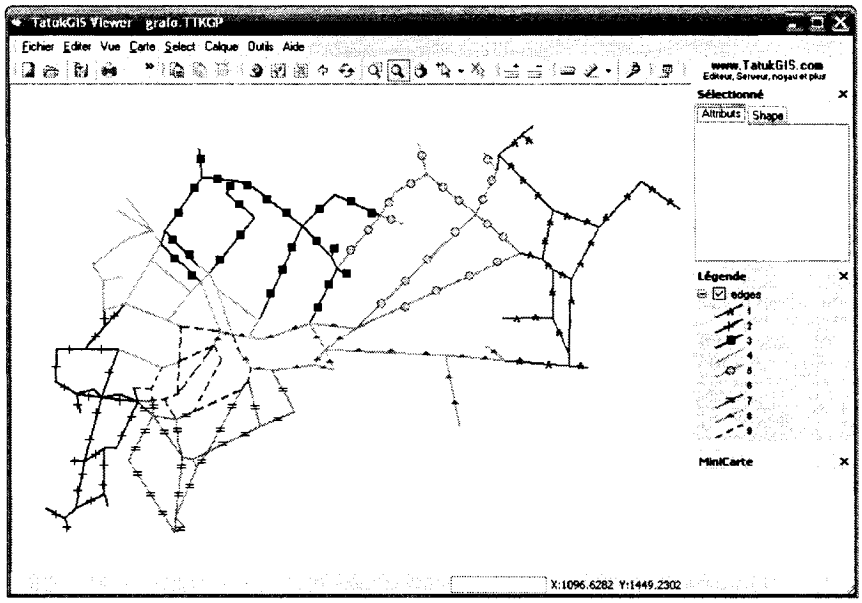


Figure 5.1. Phase 1 : Affectation des arcs à VM

Phase 2 :

La méthode utilisée à la phase 1 donne les tournées à utiliser à la phase 2. Chaque tournée a été post-optimisée avec des échanges 2-opt. Pour chaque tournée, nous avons déterminé les points de remplissage en utilisant l'algorithme présenté au chapitre 4.

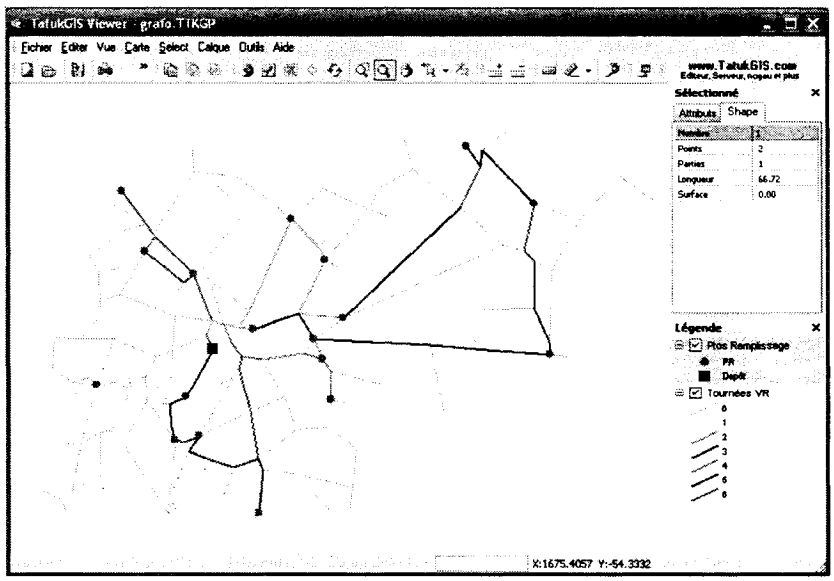


Figure 5.2. Phase 2 : localisation des points de remplissage

La Figure 5.2 présente les points de remplissage trouvés par l'algorithme pour chaque véhicule.

Phase 3 :

Le nombre total de points de remplissage trouvés à la phase 2 est 22. Tous les points ont une demande de 1, à l'exception d'un des points qui doit être visité 2 fois. Pour construire les tournées, nous avons utilisé un algorithme dérivé du VRP qui combine un algorithme de construction avec un algorithme d'amélioration. L'algorithme de construction est celui de Clarke & Wright (1964) pour le problème de tournées sur les sommets avec capacité. L'algorithme d'amélioration est le 2-opt. La Figure 5.2 présente les 6 circuits trouvés pour les véhicules de remplissage. Dans cette expérimentation, nous avons déterminé que si un point de remplissage doit être visité plus d'une fois, cela doit se faire pour le même véhicule de remplissage et dans la même tournée, c'est-à-dire que nous ne permettons pas de partition de la demande. Il faut mentionner que dans le cas où le nombre de fois qu'un point de remplissage doit être visité est supérieur à la capacité du véhicules de remplissage, la partition de la demande doit être révisée. Dans ce cas, si l'on permet de partitionner la demande, nous pouvons représenter les points à être visités comme des nœuds différents, avec une demande unitaire. De cette façon, la partition de la demande pourra être effectuée (après réduction des coûts).

Le coût total calculé est de 4257, qui correspondent à 3424 pour les véhicules de marquage et 833 pour les véhicules de remplissage. Les temps de résolution avec un ordinateur Pentium 4 et 256 mégaoctets de mémoire vive sont approximativement de 10, 3 et 1 minutes pour chacune des trois phases.

5.6 Conclusion

Le modèle mathématique développé possède un très grand nombre de variables qui est impensable à utiliser en un temps respectable pour trouver une solution exacte à un problème réel comme celui du marquage du réseau routier du MTQ. Par contre, l'heuristique proposée est une heuristique constructive qui utilise comme base des

algorithmes classiques de tournées de véhicules. Elle permet de trouver une solution en un temps nettement acceptable. Le problème décrit ici ouvre la voie à plusieurs variantes intéressantes, comme celle de la synchronisation des véhicules marqueurs et remplisseurs, qui mérite une étude plus approfondie.

CHAPITRE 6 : LIBRAIRIE D'AIDE À LA CONCEPTION DE TOURNÉES

En guise de dernière contribution de cette thèse, ce chapitre présente une librairie informatique qui a été développée pour mettre en oeuvre les algorithmes développés ici, ainsi qui la plupart des algorithmes de confection de tournées de base.

6.1 Modèle orienté-objet du marquage d'un réseau routier

Le modèle général de la librairie est basé sur les quatre grandes classes d'objets de transport, développées au sein de la modélisation orientée objet en transport (MOOT) (Trépanier, et al., 2002), c'est-à-dire :

- les objets statiques, qui possèdent une localisation fixe dans le temps et l'espace ;
- les objets dynamiques, qui sont les acteurs du transport ;
- les objets cinétiques, qui sont les descripteurs du mouvement ;
- les objets systémiques, qui sont des groupes d'objets interreliés.

Si l'on s'intéresse à l'utilisation de la géomatique comme moyen de soutien aux applications, le SIG-TOO (système d'information géographique transport orienté objet) constitue l'approche actuelle la plus pertinente à l'appréhension des divers problèmes de transport. L'amalgame de "questionneurs", de "calculateurs" et de "présentateurs" réussit à constituer interactivement une géomatique associée à ce type d'application (Trépanier & Chapleau, 2001).

Le schéma suivant (Figure 6.1) présente les interactions et les relations entre les différentes classes d'objets identifiées. Nous avons conceptualisé les objets qui sont appropriés au problème de marquage routier et qui assurent un fonctionnement efficace du système d'aide à la génération de tournées.

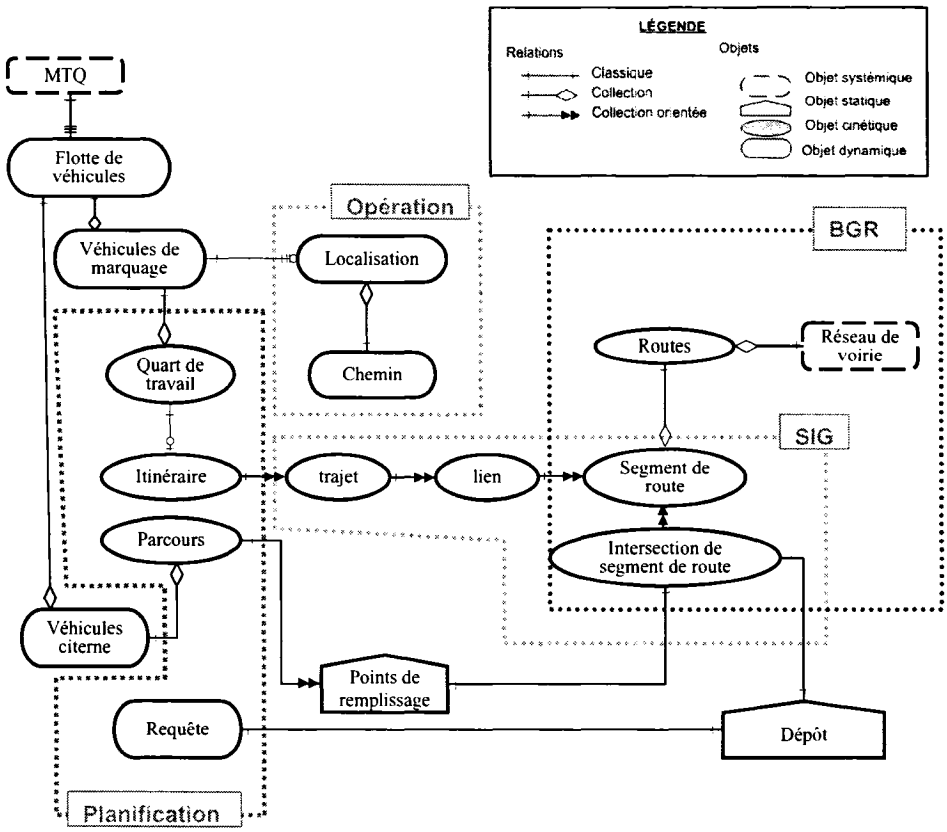


Figure 6.1. Modèle orienté-objet d'un système de tournées pour le marquage

6.1.1 Les objets SIG-T

Les objets SIG sont des objets qui modélisent la partie du réseau routier sur laquelle les travaux de marquages seront effectués. On y retrouve les objets suivants :

- l'objet <segment de route> qui sert à représenter l'entité minimale du système, où les propriétés de chaque entité jouent un rôle ;
- l'objet <lien> qui représente une collection de segments de route adjacents ayant les mêmes caractéristiques ;
- l'objet <trajet> qui représente une collection orientée de liens ;
- l'objet <intersection du segment de route> qui sert à représenter les sommets ou nœuds dans la représentation analytique du réseau.

6.1.2 Les objets BGR

Les objets BGR (Base géo-routière) sont des objets qui complètent la modélisation du réseau routier. En général, ces objets correspondent à une base de données contenant de l'information sur le réseau routier disponible. On y retrouve les objets suivants :

- l'objet <routes> qui sert à représenter l'entité maximale du système, où les propriétés de chaque entité minimale ne jouent pas de rôle ;
- l'objet <réseau de voirie> est un ensemble d'objets cinétiques spatialisés sur lesquels est habilité à circuler l'objet dynamique <flotte de véhicules>.

6.1.3 Les objets opérationnels

La composante opérationnelle est matérialisée par les objets suivants :

- l'objet <chemin> représente un ensemble de localisations visant à définir les déplacements réels d'un véhicule ;
- l'objet <localisation> caractérise le véhicule dans l'espace et dans le temps.

6.1.4 Les objets de planification

Les objets de planification représentent les résultats des méthodes et algorithmes mathématiques développés pour la planification des activités. On y retrouve les objets suivants :

- l'objet <quart de travail> représente un ensemble d'itinéraires à suivre pendant un quart de travail, par un même véhicule ;
- l'objet <itinéraire> représente une suite de trajets qu'un véhicule de marquage doit parcourir ;
- l'objet <parcours > représente une suite de points de remplissage (intersections de segments) que le véhicule de remplissage doit atteindre ;

- l'objet <requête> représente des besoins du réseau.

Il y a un problème d'intégration des données opérationnelles dans la planification des activités (les objets de planification et les objets d'opération dans la Figure 6.1). Le problème se matérialise au moment de détecter les travaux réalisés sur le réseau en recherchant les objets affectés à partir de traces captées par des GPS. Cette problématique est abordée par plusieurs chercheurs mais est loin de se résoudre dans des cas pratiques.

Les objets SIG, BGR et PLANIFICATION montrés dans la Figure 6.1 représentent à un niveau macro la série d'objets qui modélisent le cœur du problème de marquage traité dans les chapitres précédents. Ces objets peuvent aussi être utilisés dans la modélisation d'autres types de problèmes particuliers de génération de tournées de véhicules. Dans les sections suivantes, ces objets sont décrits plus précisément dans le contexte d'une librairie informatique multi-usages.

6.2 Nécessité d'une librairie pour la génération de tournées

Les applications pour la génération de tournées de véhicules utilisent une série de composantes qui sont communes entre elles. Lorsque nous développons un algorithme pour résoudre un problème, il faut bien souvent intégrer plusieurs algorithmes de base comme des calculateurs de chemins les plus courts, du calcul de composantes, ou des algorithmes plus complexes comme les problèmes de facteur rural qui sont eux-mêmes la base de plusieurs méthodes comme le problème de tournées sur les arcs avec capacité. Le temps total de développement de l'application est réduit en développant une librairie composée de différents algorithmes. Citons « Au cœur d'ActiveX et OLE », de *David Chappel* : « La réutilisation est aussi une voie vers la création de meilleurs logiciels. Aujourd'hui encore, les développeurs de logiciels en sont toujours à partir d'une certaine forme de sable et à suivre les mêmes étapes que les centaines de programmeurs qui les ont précédés. Le résultat est souvent excellent, mais il pourrait être amélioré. La création de nouvelles applications à partir de composants existants, déjà testés, a toutes chances

de produire un code plus fiable. De plus, elle peut se révéler nettement plus rapide et plus économique, ce qui n'est pas moins important.»

Une librairie est également un bon moyen de standardiser les échanges de code entre les chercheurs. Il existe des librairies pour la manipulation de graphes, par exemple <G, TL> (http://infosun.fmi.uni-passau.de/GTL/index.html) mais elles ne sont pas spécifiques pour les problèmes que nous traitons dans notre groupe de recherche, soit les problèmes de tournées de véhicules sur les arcs. Il existe aussi des librairies qui regroupent une grande quantité d'algorithmes mais qui ne sont pas réutilisables facilement (voir par exemple Lacomme, Prins & Sevaux (2003)).

6.3 Principe de fonctionnement de la librairie

Cette librairie est composée de plusieurs classes incluses dans le même espace de noms (*namespace*) "POLYROUTE". La librairie a été réalisée avec le *Framework .NET* de *Microsoft* sous Visual Basic .NET et est implanté au MTQ. Pour utiliser la bibliothèque d'algorithmes dans un projet, il faut simplement ajouter cette bibliothèque dans les références du projet.

Cet outil informatique, qui prend la forme d'une librairie orientée-objets, permet de manipuler les éléments d'un réseau routier analytique. Cet outil est muni d'un grand nombre d'algorithmes et de méthodes de confection de circuits "de base". Une fois compilée, les fonctions de la librairie sont accessibles depuis la grande majorité des logiciels implantés sur Windows, ainsi que depuis la plupart des environnements de développement.

6.3.1 Le réseau analytique

Le stockage du réseau routier prend place sur un réseau analytique avec une structure particulière. Ce réseau analytique est un réseau parallèle au réseau routier qui, dans sa structure la plus simple, correspond à un graphe mathématique. Les méthodes

mathématiques sont basées sur ce type de structure (nœud / lien). Voici la structure des données. Les champs obligatoires des tables de la base de données seront décrits ici.

Tableau 6.1. Tables de la base de données

Tables	Description
RES_NODS	Dans cette table, nous stockons des intersections de routes considérées dans le réseau parallèle. Dans la théorie des graphes, cette table définit l'ensemble des nœuds du graphe en question.
RES_ARCS	Dans cette table, nous stockons des segments de routes considérées dans le réseau parallèle. Dans la théorie des graphes, cette table définit l'ensemble d'arcs du graphe en question.
RES_ARCS_SEGMN	Dans cette table, nous définissons une relation entre les arcs du graphe mathématique et les segments du réseau routier.
Circuits	Dans cette table, nous stockons les circuits créés par l'utilisateur et les algorithmes.
Circuits_liens	Cette table contient les arcs appartenant à chaque circuit.

Le réseau analytique hérite des propriétés du réseau routier. La Figure 6.2 présente le modèle entité-relation avec les champs obligatoires des tables.

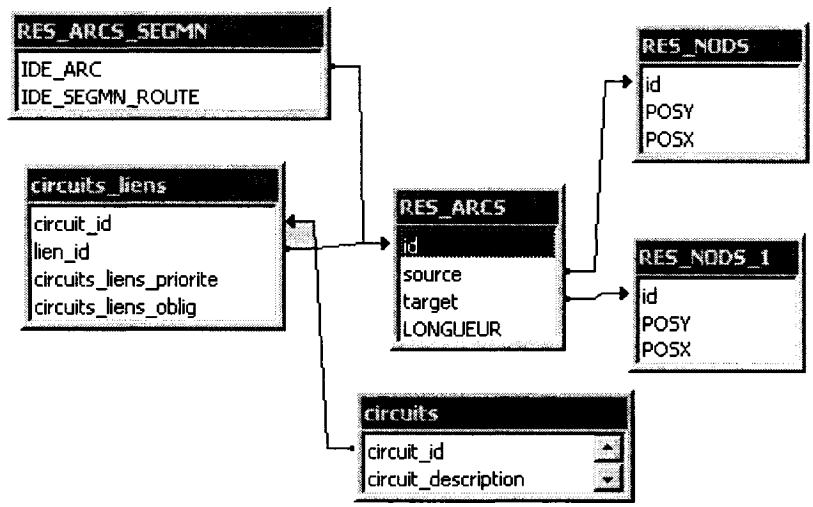


Figure 6.2 Modèle entité-relation

A noter que le champ *IDE_SEGMN_ROUTE* de la table *RES_ARCS_SEGMN* est un identificateur du segment (section de route) défini dans le SIG du MTQ.

6.4 Architecture

La documentation de la librairie se trouve en annexe. Elle a été constituée en un fichier de format HTML, utilisable directement dans l'environnement de programmation. Nous retrouvons ici les aspects les plus fondamentaux seulement.

La librairie informatique est constituée de plus de 40 classes qui permettent la manipulation de réseaux routiers en les modélisant comme *graphes.* Les classes ont été catégorisées en 4 types. Cette caractérisation se fait selon leur rôle :

- Classes du type « *AlgoXXX* » ;
- Classes du type « *ClbXXX* » ;
- Classes du type « *ColXXX* » ;
- Classes du type « *ClsXXX* ».

6.4.1 Classes du type « *Algoxxx* »

Ces classes servent à résoudre un problème spécifique de la théorie des graphes. Chaque classe est codée dans un fichier « .vb », le nom du fichier étant le même que le nom représentatif de la classe. Les fichiers de ce type de classes et les problèmes associés sont décrits dans le tableau de description générale (Tableau 6.3).

Ces classes s'utilisent à peu près toutes de la façon suivante :

- définir un objet du type *AlgoXXX* appartenant à la classe ;
- définir les paramètres d'entrée du problème en appelant les méthodes correspondantes ;
- exécuter l'algorithme codé dans la classe en appelant la méthode « *.run* ».

Une fois le problème résolu, nous disposons dans chaque classe de méthodes qui retournent des solutions trouvées. Prenons par exemple la classe « ***AlgoDijstra*** ». Cette méthode résout le problème de trouver le chemin et la distance la plus courte entre deux sommets d'un graphe. Nous disposons de la méthode suivante pour indiquer les deux sommets impliqués et l'attribut à utiliser pour l'évaluation (poids ou coût sur chaque lien) :

ClsDijstra:defParametres (nSource As *ClsNoeud*, _
nTarget As *ClsNoeud*, _
cout As *clbAttribut*)

Une fois que la méthode *ClsDijstra:run* a été appelée, nous pouvons accéder aux informations trouvées par la méthode tel que le coût et le chemin à parcourir entre les deux sommets. Les méthodes du Tableau 6.2 donnent des informations une fois que nous avons exécuté l'algorithme en question.

Tableau 6.2. Méthodes de sortie de la classe AlgoDikstra

Méthode de sortie	Description
shortDistance() As *Single*	Distance entre les deux sommets.
distance(n As *ClsNoeud*) As *Single*	Distance du sommet origine à un autre sommet.
predNode(n As *ClsNoeud*) As *ClsNoeud*	Noeud prédécesseur à un nœud sur le chemin le plus court.
predEdge(n As *ClsNoeud*) As *ClsLien*	Lien prédécesseur à un nœud sur le chemin le plus court.
reached(n As *ClsNoeud*) As *Boolean*	Vrai si l'algorithme a trouvé la distance la plus courte jusqu'à ce sommet.
shortPath() As *clsChemin*	Chemin pour aller du sommet origine au sommet destination.

6.4.2 Classes du type « *ClbXXX* »

Ce type de classe permet de manipuler des éléments de base habituellement utilisés dans les autres classes. Chaque classe est codée dans un fichier « .vb », le nom du fichier étant le même que le nom représentatif de la classe. Les fichiers de ce type de classe et

une brève description se retrouvent dans le tableau de description générale (Tableau 6.3).

6.4.3 Classes du type « *ColXXX* »

Ce type de classes permet de manipuler des ensembles d'objets (collections). Il s'agit d'une spécialisation de la classe *Collection* de Visual Basic. Chaque classe est codée dans un fichier «*.cls* ». De la même manière que les classes « C*lbxxx* », elles sont utilisées à l'intérieur des autres classes. Les fichiers de ce type de classes et une brève description sont également disponibles dans tableau de description générale (Tableau 6.3).

6.4.4 Classes du type « *ClsXXX* »

Ces classes sont le cœur de la librairie. Chaque classe est codée dans un fichier « .vb », avec un nom représentatif, se référer au tableau de description général (Tableau 6.3).

La modélisation est basée sur deux classes d'objets :

- l'objet *ClsLien* représentant un arc du graphe (réseau) ;
- l'objet *ClsNoeud* représentant un nœud du graphe.

Deux collections permettent de traiter les deux groupes d'objets :

- la collection *ClsLiens* représente une collection d'objets *ClsLien* ;
- la collection *ClsNoeuds* représente une collection d'objets *ClsNoeud* ;

La classe *ClsGraphe* est l'instanciation objet des graphes qui regroupent les classes *ClsNoeuds* et *ClsLiens*.

La classe *clsTroncon* est une spécialisation de la classe *ClsLien.* Cette classe permet d'associer des attributs à un lien donné. La classe *ClsChemin* est une collection d'objets *ClsTroncon* représentant une séquence d'arcs et il est possible de répéter un arc dans la séquence. Ceci n'est pas possible dans la collection *ClsLiens*. La collection *clsChemins*

représente une collection d'objets *ClsChemin*. La classe *clsTournee* peut être utilisée pour la création des parcours.

6.5 Hiérarchie des classes

La hiérarchie des classes est présentée dans le Figure 6.3.

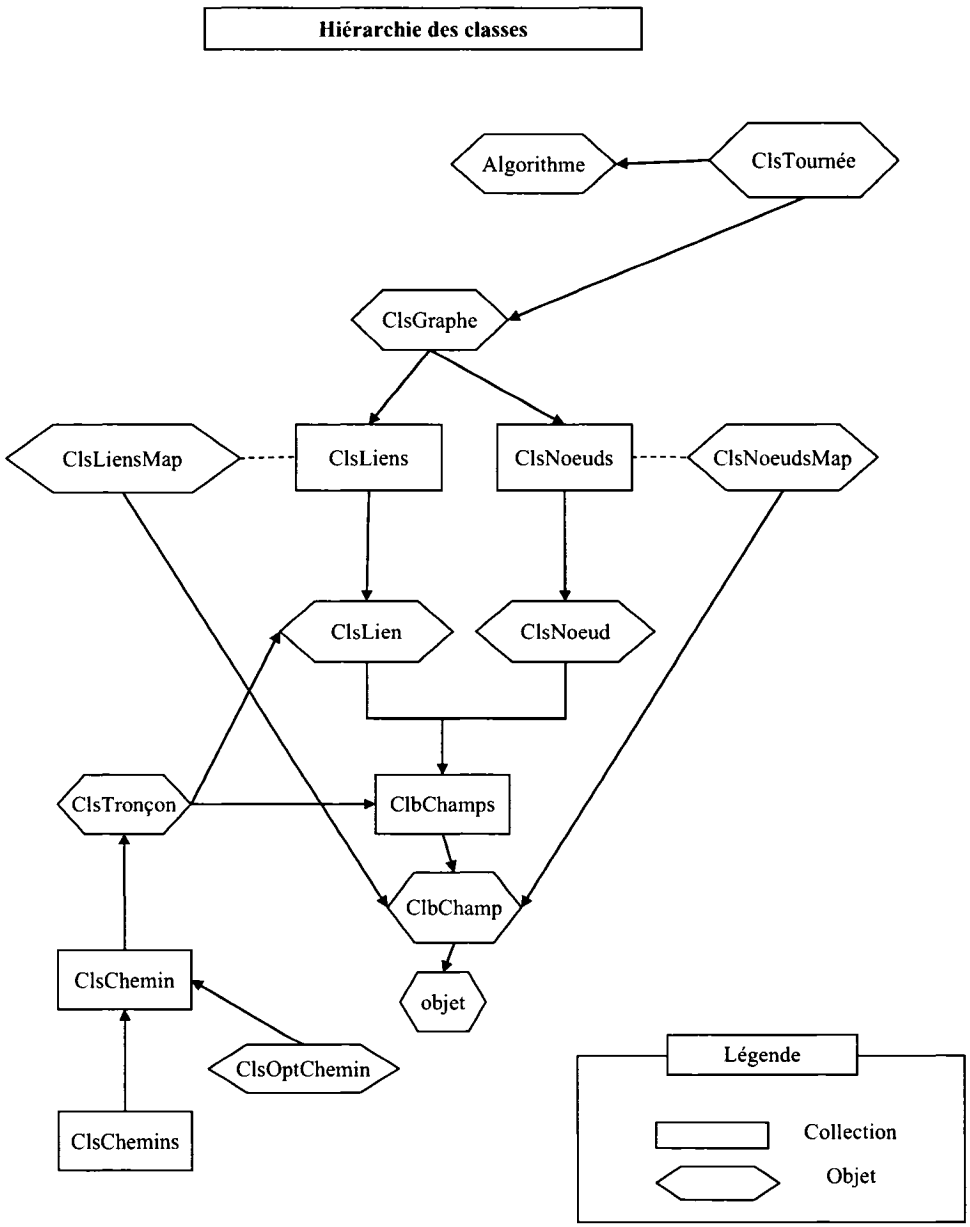


Figure 6.3. Hiérarchie des classes

La classe *ClbAttribut* sert à l'utilisation des caractéristiques additionnelles des liens et nœuds. La structure d'héritage de cette classe est montrée dans la Figure 6.4.

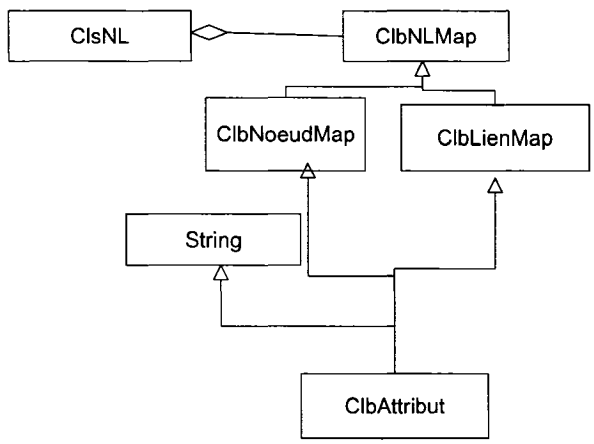


Figure 6.4. Héritage de la classe ClbAttrib

La Figure 6.5 présente la structure d'héritage des classes *ClsLiens* et *ClsNoeuds*.

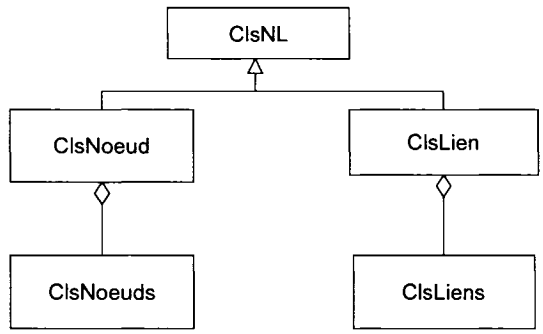


Figure 6.5. Héritage des classes *ClsLiens* et *ClsNoeuds*

6.6 Description des classes

Nous présentons ci-après au Tableau 6.3 la description générale des classes développées. Rappelons que la librairie est décrite au complet dans un fichier en format HTML et MSHELP.

Tableau 6.3. Description de classes

Class	Description
AlgoArbor	Arborescence de recouvrement de coût minimum.
AlgoBellman	Recherche d'un chemin le plus court dans un graphe.
AlgoCarp	Algorithme de tournées avec capacité.
AlgoDijkstra	Algorithme de Dijkstra (chemin le plus court).
AlgoMaxFlotMinCout	Recherche d'un flot de coût minimal dans un graphe.
AlgoParcoursPC	Recherche d'un parcours eulérien.
AlgoParcoursPR	Recherche d'un parcours passant au moins une fois dans chaque arc obligatoire.
AlgoTSP	Construction de tournées sur les nœuds (problème du commis-voyageur, TSP).
ClbAttribut	Manipule les attributs des nœuds ou des liens.
ClbChamp	Champ ou caractéristique d'un élément d'un graphe.
ClbChamps	Ensemble d'objets de la classe *ClbChamp*.
ClbMapLiens	Affecte des valeurs aux liens.
ClbMapNL	Classe de base pour l'assignation de valeurs aux liens ou aux nœuds.
ClbMapNoeuds	Affecte des valeurs aux nœuds.
ClsChemin	Collection d'objets de la classe *ClsTroncon* qui représentent un chemin (*path*) fermé.
clsChemins	Collection d'objets de la classe *ClsChemin*, c-à-d collection de chemins.
ClsGraphe	Classe représentant un graphe mathématique.
ClsLien	Classe pour la manipulation des objets « lien » d'un graphe mathématique.
ClsLiens	Collection de *ClsLien*.
clsList	Liste multi-usage.
ClsNL	Classe de base pour des objets élémentaires d'un graphe tels que les nœuds et les liens.
ClsNoeud	Classe pour la manipulation des objets « nœud », (sommets) d'un graphe mathématique.
ClsNoeuds	Collection de *ClsNoeud*.
clsOptChemin	Manipulation des chemins, voir la classe *ClsChemin*.
clsTournee	Classe représentant une tournée.
ClsTroncon	Objet élémentaire du *ClsChemin*.
clsUtil	Librairie de fonctions partagées : pas besoin d'instancier la classe « Util ».
colColIntegers	Collection de collections d'entiers.
colColLiens	Collection de collections de liens.
colColNodes	Collection de collections de nœuds.
colGen	Collection d'éléments génériques.
colIntegers	Collection d'entiers.
colLiens	Collection de liens de *ClsLien*.
colNodes	Collection de nœuds de *ClsNoeud*.
colTroncons	Collection d'objets de la classe *ClsTroncon*.
modConstants	Module des constantes, types, variables globales et procédures communes.

6.7 Utilisation de la librairie dans un cas de génération de tournées

Cette partie présente l'interface qui a été développée sur Excel pour montrer les fonctionnalités de la librairie ainsi que pour conduire des analyses.

6.7.1 Macro *VISRES.XLA*

L'utilisation de la librairie DLL à partir du tableur Microsoft Excel se fait par le biais d'un fichier de macros complémentaires Excel nommé *VISRES.XLA*. Les fichiers de type XLA ont la particularité de se charger en parallèle à tout classeur Excel. La configuration de l'appel du XLA se fait via la boîte de dialogue située dans le menu « Outils / Macros complémentaires » du logiciel Excel.

La macro *VISRES.XLA* est codée en Visual Basic pour Applications (VBA). L'examen du code de cette macro est un bon moyen de comprendre comment utiliser la librairie DLL. Cependant, beaucoup d'éléments de code, tel que les fenêtres (*Userforms*), sont relatifs seulement à l'application Excel et ne pourraient être transposés directement dans un autre environnement de programmation.

L'appel des procédures contenues dans la macro se fait à l'aide d'une barre d'outils appelée « Gestion Réseau », qui s'affiche lors de son démarrage.

6.7.2 Classeur Excel

En plus de la macro *VISRES.XLA*, un classeur Excel est nécessaire au stockage des données du réseau routier ainsi que pour l'affichage. Ce classeur est structuré en quatre feuilles de calcul :

- « Réseau » sert à l'affichage graphique du réseau et des circuits ;
- « RES_NODS» contient les attributs des nœuds du réseau. En plus de l'identifiant (qui doit être unique), un nœud est caractérisé par des positions X et Y en coordonnées « Excel » (qui correspondent environ à des pixels) ;

- « RES_ARCS » contient les attributs des liens du réseau. Le Tableau 6.4 présente ces attributs ;
- « Paramètres » contient les paramètres du cas étudié. Le Tableau 6.5 présente ces paramètres.

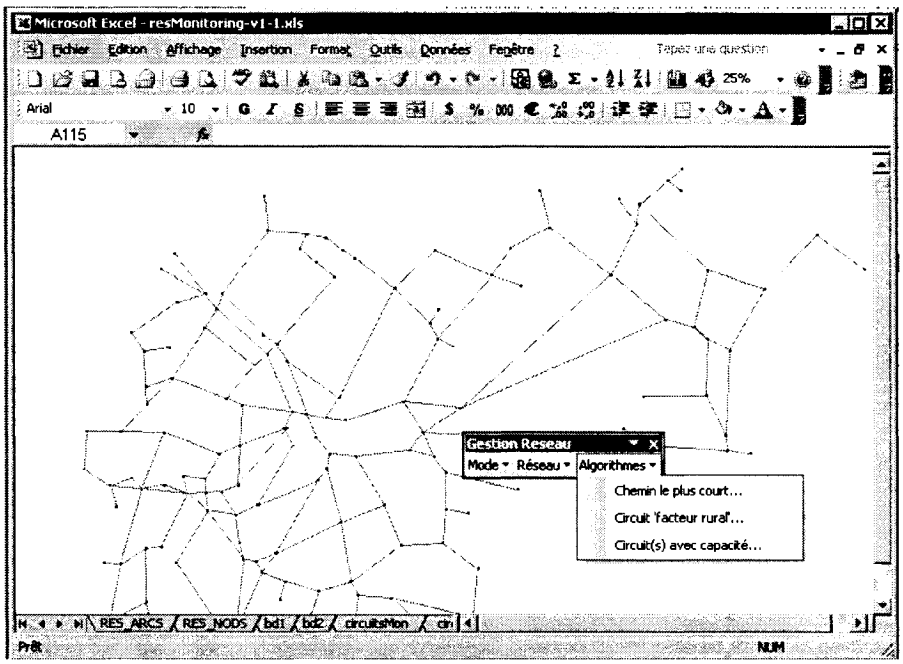


Figure 6.6. Interface Excel

Tableau 6.4. Attributs des liens dans le classeur Excel, feuille "RES_ARCS"

Champ	Contenu
id	Identifiant du lien
Noeud1	Nœud d'origine
Noeud2	Nœud de destination
Nom	Nom donné au lien
Sens	Tous les liens sont habituellement dirigés dans le réseau
Classe	Classe de route (attribut du MTQ)
Couverture	Indique si le lien a été couvert ou non lors d'une simulation
Route	Descriptif de la route associé au lien routier
longueur	Longueur du lien (km)
Oblig1	Attribut de lien obligatoire
direction	Direction générale du lien (N, S, E,W)
classeX	Attribut variable permettant de changer de paramètre d'arc obligatoire (1 = obligatoire, 0 = non-obligatoire)

Tableau 6.5. Paramètres du classeur

Paramètre	Signification
NomReseau	Nom donné au projet
Node Depot	Numéro du noeud servant de dépôt (garage)
Capacité	Capacité des véhicules (ici, en km ou selon l'attribut de coût)
Conversion	Facteur de conversion des pixels en km
Attribut Coût	Attribut de lien utilisé pour le coût
Attribut Obligatoire	Attribut de lien utilisé pour indiquer les arcs obligatoires
Attribut Demande	Attribut de lien utilisé pour quantifier la demande
Attribut Classe	Attribut de lien indiquant la classe de route
Changer Échelle	Changer l'échelle du réseau dans la feuille
Path des fichiers	Répertoire où se trouvent les fichiers (si vide, le même que le classeur actuel)
Nom fichier données	Nom du fichier de données (si vide, le classeur actuel)
Nom fichier chemins	Nom du fichier Access contenant les circuits (si vide, le fichier Access porte le même nom que le classeur actuel)
type circuits	Caractéristique du circuit (ex., monitoring, marquage)

6.7.3 Base de données Access

Une base de données Access accompagne le classeur Excel. Elle permet de stocker tous les ensembles de circuits calculés lors des simulations. Pour chaque ensemble de circuits, il y a deux tables créées dans Access : une pour stocker le nom des circuits et l'autre pour stocker la séquence de liens relative à ces circuits. Le fichier Access est entièrement édité par la macro *VISRES.XLA*, nul besoin de l'ouvrir.

6.7.4 Transposition

L'architecture propre du classeur lui confère une certaine rigidité. Ainsi, pour utiliser les exemples et les modifier, il faut conserver la structure actuelle. Pour faire d'autres simulations, il suffit de copier le classeur actuel sous un autre nom et renommer également le fichier Access (du même nom).

6.7.5 Mode d'emploi

Voici une brève description des fonctions de la barre d'outils « Gestion Réseau » et de l'interface Excel.

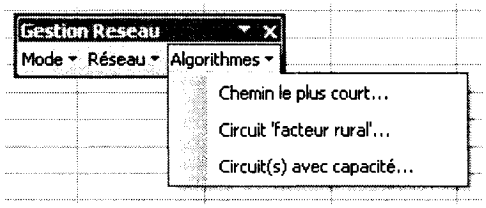


Figure 6.7: Barre d'outils "Gestion Réseau"

6.7.6 Menu « Mode »

Le menu « Mode » permet de changer le mode d'édition de l'interface :

- **Interrogation** : Dans le mode « Interrogation », on peut interroger les caractéristiques d'un lien ou d'un nœud en cliquant dessus. Le bouton  permet d'interroger le lien inverse, puisque les liens contraires sont souvent superposés dans l'interface.

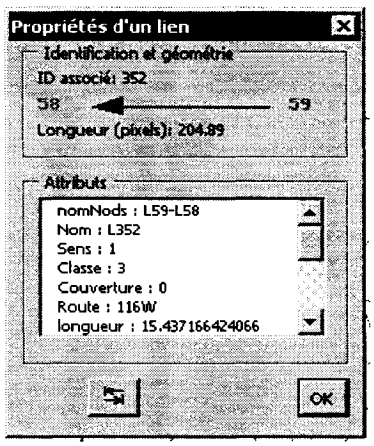


Figure 6.8: Propriétés d'un lien

- **Déplacement des nœuds** : Dans ce mode, il est possible de déplacer les nœuds du réseau (les liens suivent alors). Ce mode doit être utilisé avec soin car il risque de faire déconnecter des liens du réseau.

- **À propos de... :** Identification de l'interface.

6.7.7 Menu « Réseau »

Le menu « Réseau » sert aux fonctions relatives à la visualisation des éléments du réseau.

- **Propriétés générales** : Cette boîte de dialogue calcule sur demande les propriétés géométriques et mathématiques du réseau.
- **Redessiner le réseau** : Cette fonction redessine complètement les objets réseau de l'interface à partir des attributs stockés.
- **Affichage des tronçons obligatoires** : Affiche les tronçons obligatoires du réseau après choix de l'attribut associé.
- **Affichage des circuits** : Cette boîte de dialogue affiche le circuit sélectionné dans la liste. En cliquant sur  (Figure 6.9), on accède à une boîte de dialogue d'édition de chaque circuit (Figure 6.10). Dans la figure on peut voir les différentes fonctions d'édition qui sont offertes.

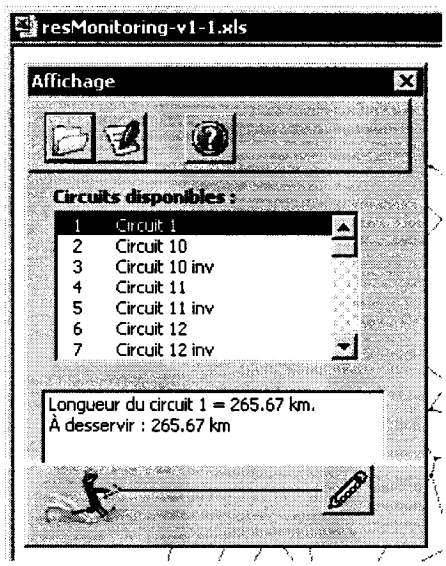


Figure 6.9. Boite de dialogue d'affichage des circuits

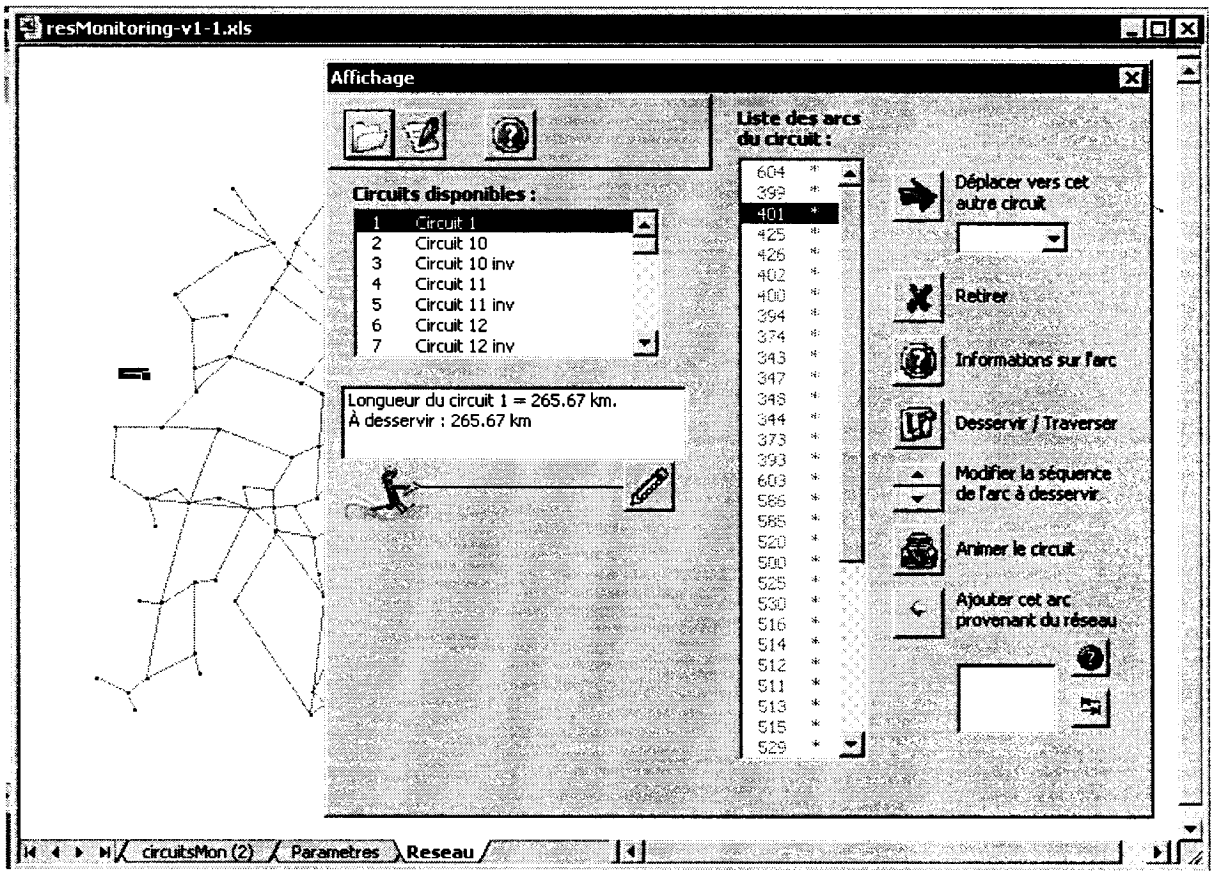


Figure 6.10. Affichage d'édition de circuits

6.7.8 Menu « Algorithmes »

La Figure 6.7 montre les options développés dans cet outil, il y a trois algorithmes utilisés : le chemin le plus court (utilise la classe *AlgoDijstra*), circuit de facteur rural (utilise la classe *AlgoParcoursPR*) et génération de circuits avec capacité (*AlgoCARP*). La Figure 6.11 montre l'interface pour sélectionner les attributs et paramètres dans l'appel de la fonction de génération de circuits avec capacité. On peut voir dans la figure les différents paramètres à sélectionner et une fenêtre pour sélectionner (filtrer) les arcs obligatoires dans la construction de circuits.

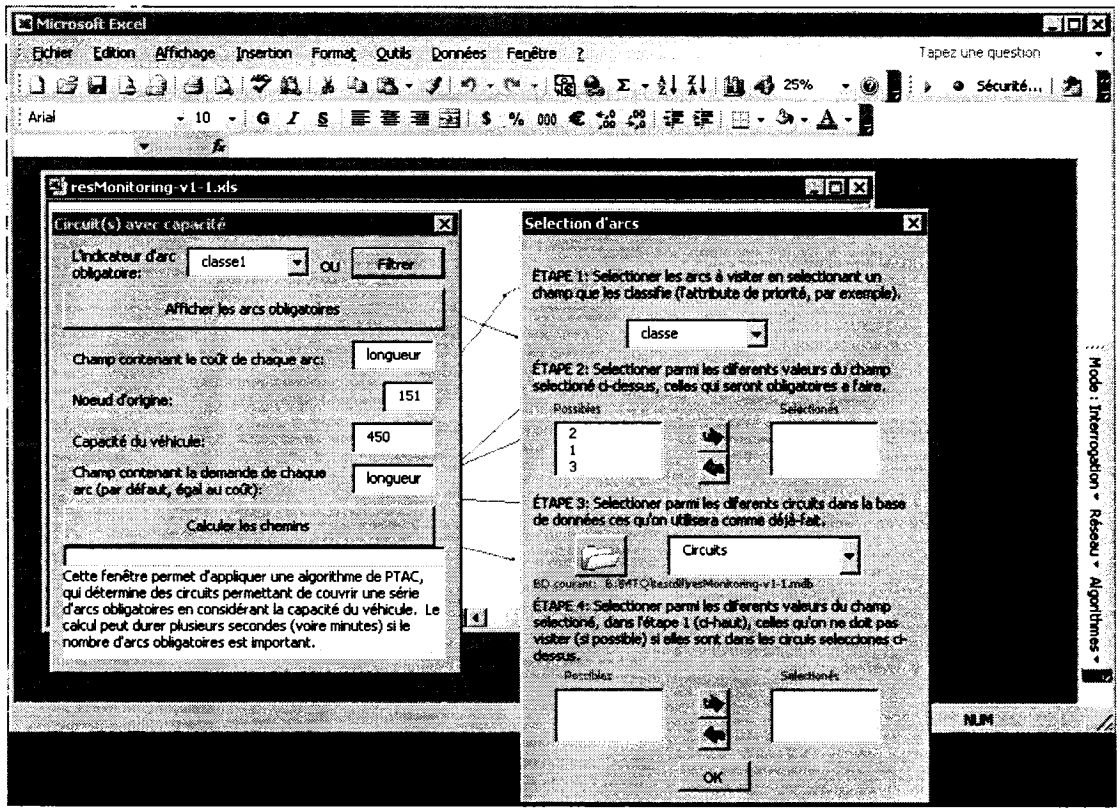


Figure 6.11. Fenêtre de création de circuits avec capacité

CHAPITRE 7 : DISCUSSION GÉNÉRALE ET CONCLUSION

Cette thèse présente un nouveau problème qui est celui marquage routier. Le problème a été étudié en considérant les aspects mathématique et informationnel. Dans les chapitres 3, 4 et 5, nous avons présenté et modélisé trois variantes du problème à l'aide de la programmation mathématique en nombres entiers. Pour chaque situation, des méthodes de résolution ont été données. Le chapitre 3 a présenté une méthode de résolution exacte qui a été augmentée par les situations décrites dans les chapitres 4 et 5. Cette méthode de résolution exacte est utile pour résoudre des problèmes de petite taille. Pour résoudre des problèmes de taille plus grande, des méthodes de résolution heuristique ont été présentées aux chapitres 4 et 5. Le Tableau 7.1 présente un résumé des variantes du problème de marquage traitées dans cette thèse.

Tableau 7.1. Résumé des problèmes traités

Chapitre	Véhicules marqueurs	Véhicules de remplissage	Capacité du véhicule de marquage	Nombre maximal de points de remplissage à visiter avant de retourner au dépôt	Nombre maximal de fois que chaque VM doit être rempli
3	1	1	Q	1	P
4	1	1	Q	infini	P
5	V	K	Q	F	P

Pour résoudre de façon exacte les problèmes traités dans cette thèse (chapitres 3, 4 et 5), nous avons utilisé une méthode classique de plans de coupe avec des stratégies pour améliorer le temps de résolution. Ces stratégies bénéficient de la modélisation utilisée. Cette méthode a été choisie parce qu'elle est plus facile à mettre en œuvre. Une méthode de *Branch and Cut* peut aussi être utilisée mais avec un temps de développement nettement supérieur. Rappelons que, de toute façon, compte tenu de la complexité du modèle, seules de petites instances peuvent être résolues en un temps raisonnable et nous devons passer rapidement aux heuristiques.

Nous avons réussi à développer des heuristiques constructives pour les problèmes de marquage de réseaux routiers. Le chapitre 4 présente une heuristique pour le cas d'un seul véhicule de remplissage et un seul véhicule de marquage. Cette heuristique est une composante de la procédure développée au chapitre 5. Cette heuristique a été testée sur des problèmes générés aléatoirement et il semble donner de bons résultats. En outre, l'heuristique développée au chapitre 5 a été testée sur un graphe représentant le réseau routier réel de la région du Québec étudiée. Cette heuristique résout théoriquement le problème, mais il reste à être opérationnalisé. Cette implantation n'a pas été possible à cause de contraintes internes au MTQ. Entre autres, la base de données géoroutière du ministère n'est pas prête, pour le moment, à inclure ce genre d'algorithmes.

Étant donné que les problèmes traités sont de nouveaux problèmes dans la littérature, les heuristiques présentées ici sont les premières ébauches pour ces problèmes. Elles semblent efficaces mais n'ont évidemment pas pu être comparées. Le travail ici présenté servira donc de base de comparaison pour créer de nouvelles heuristiques peut-être plus performantes.

Une autre contribution de cette thèse est de présenter un cadre de modélisation informationnelle des problèmes liés à l'entretien des réseaux routiers qui inclut les modèles présentés aux chapitres 3, 4 et 5. Dans ce contexte, nous avons réussi à développer une librairie informatique qui met en oeuvre les algorithmes développés ainsi que la plupart des algorithmes de confection de tournées de base. Cette librairie est un outil important, non seulement pour des chercheurs, mais aussi pour les planificateurs. L'intégration de cette libraire dans des projets de confection de tournées est relativement facile à faire. Une démonstration du prototype développée dans le cadre des tournées de surveillance routière a été présentée au chapitre 6. En outre, cette librairie a été implantée avec succès dans l'environnement intranet du MTQ. Elle a été utilisée pour faire un service web qui regroupe les principales fonctions de la librairie.

Dans cette thèse, nous avons montré, de façon générale, que la planification de la réfection du marquage de la chaussée sur un réseau donné consiste à déterminer quelles

routes doivent être peintes et à quel moment le faire. Nous avons déterminé que le processus de marquage peut être divisé en étapes complémentaires. Dans un premier temps, au niveau stratégique, la période de réfection des marquages est planifiée. La deuxième étape, au niveau de planification saisonnière, consiste à définir les interventions exactes à faire durant une certaine période. À la fin de cette étape, les circuits du véhicule de marquage (VM) sont établis. Finalement, à la dernière étape, au niveau opérationnel, le suivi des opérations est effectué. À cette étape, les circuits sont modifiés dynamiquement afin de réagir aux changements de dernière minute et pour ajuster les circuits selon les écarts entre le travail planifié et le travail effectué.

Étant donné la complexité du problème de marquage routier, nous nous sommes concentrés sur l'étape de planification saisonnière (Étape 2 de la Figure 0.1). Cette étape est elle-même très complexe et nous avons dû mettre beaucoup d'efforts sur les développements en recherche opérationnelle, en modélisation, en implantation informationnelle et en intégration de données. Les problèmes étudiés dans cette thèse ouvrent la voie à plusieurs problèmes intéressants dans le domaine de l'entretien des réseaux routiers et d'autres domaines comme le réapprovisionnement d'avions en vol. Des recherches supplémentaires sont nécessaires à la fois dans la modélisation et dans la résolution de tels problèmes. Parmi les nouvelles perspectives de recherches, il faut considérer l'ajout de nouvelles contraintes opérationnelles telles que la hiérarchie du réseau, les fenêtres de temps pour la synchronisation des véhicules, les interdictions de virage, etc.

RÉFÉRENCES BIBLIOGRAPHIQUES

AMAYA, A., LANGEVIN, A. & TREPANIER, M. (2007). The capacitated arc routing problem with refill points. *Operations Research Letters.* 35 : 1. 45-53.

AMAYA, A., TRÉPANIER, M. & LAPIERRE, S. (2003). Outil léger de planification de tournées d'autobus d'écoliers pour un transporteur de petite taille. *Communication présentée au 38e congrès de l'Association québécoise du transport et des routes.*

ANSELIN, L., DODSON, R.F. & HUDAK, S. (1993). Linking GIS and Spatial Data Analysis in Practice. *Geographical Systems.* 1 : 1. 3-23.

ASSAD, A.A. & GOLDEN, B.L. (1995). Arc ruting methods and applications. In M.O. BALL, T.L. MAGNANTI, C.L. MONMA & G.L. NEMHAUSER, *Network Routing, Handbooks in Operations Research and Management Science.* (pp. 375-483). Amsterdam, The Netherlands. North-Holland.

BALDACCI, R. & MANIEZOO, V. (2004). Exact methods based on node routing formulations for arc routing problems. *Department of Computer Science, University of Bologna.* 18 p. UBLCS-2004-10.

BEASLEY, J.E. (1983). Route first-cluster second methodes for vehicle routing. *Omega.* 11 : 403-408.

BELENGUER, J.-M. (2006). The capacitated arc routing problem; Data Sets. Page consultée le 15 Juin 2006, tiré de http://www.uv.es/~belengue/carp.html.

BELENGUER, J.-M. & BENAVENT, E. (1998). The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization & Applications.* 10 : 2. 165-187.

BELENGUER, J.-M., BENAVENT, E., LACOMME, P. & PRINS, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem: Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems. *Computers & Operations Research.* 33 : 12. 3363-3383.

BENAVENT, E., CAMPOS, V.C.A. & MOTA, E. (1990). The Capacitated Arc Routing Problem A Heuristic Algorithm. *Qüestiió.* 1-3 : 107-122.

BENAVENT, E., CAMPOS, V.C.A. & MOTA, E. (1992). The Capacitated Arc Routing Problem. Lower Bounds. *Networks.* 22 : 669-690.

BEULLENS, P., MUYLDERMANS, L., CATTRYSSE, D. & VAN OUDHEUSDEN, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research.* 147 : 3. 629-643.

BLAIS, M. & LAPORTE, G. (2002). Exact Solution of the Generalized Routing Problem through Transformations. *Centre de recherche sur les transports (C.R.T.).*

BOULIANE, J. & LAPORTE, G. (1992). Locating postal relay boxes using a set covering algorithm. *American Journal of Mathematical and Management Sciences.* 12 : 65-74.

CAMPBELL, J.F. & LANGEVIN, A. (1995). Operations management for urban snow removal and disposal. *Transportation Research A.* 29A : 5. 359-370.

CHAPLEAU, L., FERLAND, J.-A., LAPALME GUY & ROUSSEAU JEAN-MARC . (1984). A Parallel Insert Method for the Capacitated Arc Routing Problem. *Operations Research letters.* 3 : 2. 95-99.

CHRISTOFIDES, N. (1973). The optimum traversal of a graph. *Omega.* 1 : 719-732.

CHRISTOFIDES, N., CAMPOS, V., CORBERAN, A. & MOTA, E. (1986). An Algorithm for the Rural Postman Problem on a Directed Graph. *Mathematical Programming.* Study 26 : 155-166.

CHULMIN, J. (2000). Design of an Intelligent Geographic Information System for Multi-Criteria Site Analysis. *URISA Journal.* 12 : 3. 5-17.

CLARKE, G. & WRIGHT, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research.* 12 : 568-581.

CORBERÁN, A., ROMERO, A. & SANCHIS, J.M. (2003). The Mixed General Routing Problem Polyhedron. *Mathematical Programming.* 96 : 1. 103-137.

CORBERÁN, A. & SANCHIS, J.M. (1994). A polyhedral approach to the rural postman problem . *European Journal of Operational Research.* 79 : 95-114 .

CORDEAU, J.-F., GENDREAU, M., HERTZ, A., LAPORTE, G. & SORMANY, J.-S. (2005). New Heuristics for the Vehicle Routing Problem. In A. LANGEVIN & D. RIOPEL, *Logistics Systems: Design and Optimization.* (pp. 279-298). Springer.

DE ROSA, B., IMPRONTA, G., GHIANI, G. & MUSMANNO, R. (2002). The Arc Routing and Scheduling Problem with Transshipment. *Transportation Science.* 36 : 3. 301-313.

DROR, M.(2000). *ARC Routing : Theory, Solutions and Applications.* Boston, USA : Kluwer Academic Publishers.

DROR, M. & LANGEVIN, A. (2000). Transformations and exact node routing solutions by column generation. In M. DROR, *Arc Routing: Theory, Solutions, and Applications.* (pp. 275-326). Kluwer Academic Publishers.

DUEKER, K.J. & BUTLER, A.J. (1998). GIS-T Enterprise Data Model with Suggested Implementation Choices. *Journal of the Urban and Regional Information Systems Association.* 10 : 1.

DUEKER, K.J., BENDER, P. & ZHANG, J. (2000). Sharing Transportation GIS Data. Catalog Number PR114 : 17.

EASA, S.M. (1991). Note on an optimization model for pavement marking systems. *European Journal of Operational Research.* 51 : 1. 136-140.

EDMONDS, J. & JOHNSON, E.L. (1973). Matching, Euler tours and the Chinese postman problem. *Mathematical Programming.* 5 : 88-124.

EGLESE, R.W. (1994). Routing Winter Gritting Vehicles. *Discrete Applied Mathematics.* 48 : 3. 231-244.

EGLESE, R.W. & LETCHFORD, A.N. (2000). Polyhedral theory for Arc Routing Problems. In M. DROR. *Arc Routing: Theory, Solutions, and Applications.* (pp. 199-230). Kluwer Academic Publishers.

EGLESE, R.W. & LI, L.Y.O. (1996). A tabu search Based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline . *Meta-Heuristic: Theory and Applications.* 633-649.

EISELT, H.A., GENDREAU, M. & LAPORTE, G. (1995). Arc Routing Problems, Part 1: The Chinese Postman Problem. *Operations Research.* 43 : 231-242.

EISELT, H.A. & LAPORTE, G. (1995). Objectives in location problems. In *Facility Location, a Survey of Applications and Methods.* Springer Series in Operations Research.

ESRI. (2003a). Page consultée le 15 Juin 2006, tiré de http://www.esri.com/base/gis/abtgs/what_gi.html.

ESRI. (2003b). Glossary of GIS Terms. Page consultée le 15 Juin 2006, tiré de http://www.esri.com/library/glossary/glossary.html.

FHA. (2001). Manual on Uniform Traffic Control Devices (MUTCD). Page consultée le 15 Juin 2006, tiré de http://mutcd.fhwa.dot.gov/kno-millennium_12.28.01.htm.

FICHER, M. & JAIKUMAR, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks.* 11 : 109-124.

FLETCHER, D. (2000). Geographic information systems for transportation: a look forward. In Transportation in the new millenium: state of the art and future directions. *Transportation Research Board.*

FORD, L.R. & FULKERSON, D.R. (1963). Flows in Networks. Princeton, NJ : Princeton Univ. Press.

FOSTER, E., BARNARD, A.J. & ZHAO, L. (1994). PoeT: Object Engineering in Public Transport. 2003.

FREDERICKSON, G.N. (1979). Approximation algorithms for some postman problems. *Journal of the Association for Computing Machinery.* 26 : 3. 538-554.

FREDERICKSON, G.N., HECHT, M.S. & KIM, C.E. (1978). Approximation algorithms for some routing problems. *SIAM Journal on Computing.* 7 : 2. 178-193.

FSC LTD. (2003). The Object-Oriented Advantage. Page consultée le 15 Juin 2006, tiré de http://www.firststep.com.au/education/solid_ground/oo_dev.html.

GHIANI, G., GUERRIERO, F., LAPORTE, G. & MUSMANNO, R. (2004). Tabu Search Heuristics for the Arc Routing Problem with Intermediate Facilities under Capacity and Length Restrictions. *Journal of Mathematical Modelling and Algorithms.* 3 : 3. 209-223.

GHIANI, G., IMPROTA, G. & LAPORTE, G. (2001). The capacitated arc routing problem with intermediate facilities. *Networks.* 37 : 3. 134-143.

GHIANI, G. & LAPORTE, G. (1999). Eulerian Location Problems. *Networks.* 34 : 4. 291-302.

GHIANI, G. & LAPORTE, G. (2000). A branch and cut algorithm for the undirected rural postman problem. *Mathematical Programming.* 87 : 3. 467-481.

GHIANI, G. & LAPORTE, G. (2001). Location-Arc Routing Problems. *Opsearch.* 38 : 151-159.

GOLDEN, B.L., DEARMON, J.S. & BAKER, E.K. (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research.* 10 : 1. 47-59.

GOLDEN, B.L. & WONG, R.T. (1981). Capacitated arc routing problem. *Networks.* 11 : 305-315.

GOODCHILD, M.F. (2000). GIS and transportation: Status and challenges. *GeoInformatica.* 4 : 127-139.

GOODCHILD , M.F. (1992). Integrating GIS and spatial data analysis : problems and possibilities. *International Journal of Geographical Information Science* . 6 : 407-423.

HAKIMI, S.L. (1965). Optimum Distribution of Switching Centers in a Communication Network and some Related Graph Theoretic Problems. *Operations Research.* 13 : 462-475.

HAWKINS, H.G., PARHAM, A.H. & WOMACK, K.N. (2002). Feasibility study for an All-white pavement marking System. *TRB - Transportation Research Board : CHRP report 484.*

HERTZ, A., LAPORTE, G. & MITTAZ, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Operations Research.* 48 : 1. 129-135.

HERTZ, A. (2005). Recent trends in arc routing. In M.C. GOLUMBIC & I. BEN-ARROYO HARMAN, *Graph Theory, Combinatorics and Algorithmics: Interdisciplinary Applications*. (pp. 215-236). Springer.

HERTZ, A. & MITTAZ, M. (2000). Heuristic Algorithms. In M. DROR. *Arc Routing: Theory, Solutions, and Applications*. (pp. 327-386). Kluwer Academic Publishers.

HERTZ, A. & MITTAZ, M. (2001). A Variable Neighborhood Descent Algorithm for the Undirected Capacitated Arc Routing Problem. *Transportation Science.* 35 : 425-434.

HIRABAYASHI, R., SARUWATARI, Y. & NISHIDA, N. (1992). Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research.* 9 : 2. 155–175.

JANSEN, K. (1993). Bounds for the general capacited routing problem. *Networks.* 23 : 165-173.

KIUCHI, M., SHINANO, Y., HIRABAYASHI, R. & SARUWATARI Y. (1995). An exact algorithm for the capacitated arc routing problem using Parallel Branch and Bound method. *Abstracts of the 1995 Spring National Conference of the Operational Research Society of Japan.* p. 28–9 .

KOUSKOULAS, V. (1988). An optimization model for pavement marking systems. *European Journal of Operational Research.* 33 : 3. 298-303.

LABBÉ, M., LAPORTE, G. & RODRIGUEZ-MARIN, I. (1998). Path, tree and cycle location. *Fleet Management and Logistics.* 187-204.

LACOMME, P., PRINS, C. & RAMDANE-CHERIF, W. (2002). General Arc Routing Problems Solved by a Cutting Plane Algorithm and a Genetic Algorithm. *IFAC'2002, 15th Triennial World Congress of the International Federation of Automatic Control.*

LACOMME, P., PRINS, C. & RAMDANE-CHERIF, W. (2004). Competitive Memetic Algorithms for Arc Routing Problems. *Annals of Operations Research.* 131 : 1 - 4. 159-185.

LACOMME, P., PRINS, & SEVAUX.(2003). *Algorithmes de graphes* . Paris, France : Eyrolles.

LAPORTE, G. (1988). Location - Routing Problems. *Studies in Management Science and Systems.* 16 : 163-197 .

LENSTRA, J.K. & RINNOOY, K. (1976). On General Routing Problem. *Networks.* 6 : 273-280.

LEVY, L. & BODIN, L.D. (1989). The Arc Oriented Location Routing Problem . *INFOR, Canadian Journal of Operational Research and Information Processing.* 27 : 1. 74-94.

LI, L.Y.O. (1992). *Vehicle Routeing for Winter Gritting.* Ph.D. Thesis, Lancaster University, Department of OR & OM, United Kingdom.

LI, L.Y.O. & EGLESE, R.W. (1996). An interactive Algorithm for Vehicle Routeing for Winter - Gritting. *Operational Research Society.* 47 : 217-228 .

LOTAN, T., CATTRYSSE, D. & OUDHEUSDEN, D.V. (1996). Winter Gritting in the Province of Antwerp : a Combined Location and Routing Problem. *Belgian Journal of Operational Research, Statistics and Computer Science.* 36 : 2-3. 141-157.

MARTINEZ, J.C. (1996). *STROBOSCOPE: State and Resource based Simulation of Construction Processes* . 540. p. Ph.D. Thesis, University of Michigan, USA.

MARZOLF, F. (2003). *Modélisation informationnelle et mathématique des opérations de surveillance du réseau routier.* Mémoire de maîtrise, École polytechnique de Montréal, Canada.

MITTAZ, M. (2000). *Problèmes de cheminements optimaux dans les réseaux avec contraintes associées aux arcs.* 243 p. Thèse de doctorat, École polytechnique fédérale de Lausanne, Suisse.

MTQ. (2002a). Réseau routier. Page consultée le 15 June 2006a, tiré de http://www.mtq.gouv.qc.ca/fr/reseau/classes.asp.

MTQ. (2002b). Tome V Signalisation routière. *Collection : Normes – Ouvrages routiers.*

MTQ. (2006). Réseau routier. Page consultée le 15 Juin 2006, tiré de http://www.mtq.gouv.qc.ca/fr/securite/routiere/actions/entretien.asp.

MUYLDERMANS, L., BEULLENS, P., CATTRYSSE, D. & VAN OUDHEUSDEN, D. (2001). The *k*-opt approach for the general routing problem. *Center for Industrial Management, Katholieke Universiteit Leuven.* Working paper 01/18.

MUYLDERMANS, L., CATTRYSSE, D. & VAN OUDHEUSDEN D. (2002). The p Dead-Mileage Problem . *Center for Industrial Management, Katholieke Universiteit Leuven.* Working Paper 02/02.

ORLOFF, C.S. (1974). A fundamental problem in Vehicle Routing. *Networks.* 4 : 33-64.

PEARN, W.L. (1991). Augment-insert algorithms for the capacitated arc routing problem. *Computers and Operations Research.* 18 : 2. 189-198.

POLACEK, M., DOERNER, K.F., HARTL, R.F. & MANIEZZO, V. (2006). *A Variable Neighborhood Search for the Capacitated Arc Routing Problem with Intermediate Facilities. Department of business Administration, University of Vienna, Austria.* ID: 2132.

POPINEAU, F. (2000). Introduction à la conception Orientée Objet et à UML. Page consultée le 15 June 2006, tiré de http://www.site.uottawa.ca/~bochmann/SEG2501/ Notes/UML_Presentation.pdf.

RAFANELLI, M. (1998). A graphical interface to define and store data on transportation using an object-oriented Geographic Information System. *8th World Conference on Transportation Research.*

RAMDANE-CHERIF, W. (2002). *Problčmes d'optimisation en tournées sur arcs.* 186 p. Thčse de doctorat, Université de Technologie de Troyes, France.

RIZZI, M. & GUICHOUX, B. (1997). Systčme d'information objet pour l'exploitation des réseaux de surface. *RATP, Paris.*

SALIM, M., TIMMERMAN, M., STRAUSS, T. & EMCH, M. (2002a). Artificial Intelligence-Based Optimization of Management of Snow Removal. *Technical Report Center for Transportation Research and Education (CTRE) at Iowa State University.*

SALIM, M.D., STRAUSS, T. & EMCH, M. (2002b). A GIS-Integrated Intelligent System for Optimization of Asset Management for Maintenance of Roads and Bridges . *Lecture Notes in Computer Science.* 2358 : 628-630.

TARANTILIS, C.D., DIAKOULAKI, D. & KIRANOUDIS, C.T. (2002). Combination of geographical information system and efficient routing algorithms for real life distribution operations. *European Journal of Operational Research.* In Press, Corrected Proof.

TOTH, P. & VIGO.(2001). *The Vehicle Routing Problem.* Philadelphia, USA : Society for Industrial and Applied Mathematics.

TRÉPANIER, M. (1999). *Modélisation totalement désagrégée et orientée-objet appliquée aux transports urbains.* Thèse de doctorat, École polytechnique de Montréal, Canada.

TRÉPANIER, M. & CHAPLEAU, R. (2001). Linking Transit Operational Data to Road Network with a Transportation Object-Oriented GIS. *Urban and Regional Information Systems Association Journal, Park Ridge, IL.* 13 : 2. 23-27.

TRÉPANIER, M., CHAPLEAU, R. & ALLARD, R. (2002). Geographic information system for transportation operations: models and specificity. *30e conférence annuelle de la Société canadienne de génie civil.* Pgs : 559-568.

ULUSOY, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research.* 22 : 3. 329-337.

WOHLK, S. (2003). Simulated Annealing for the Capacitated Arc Routing Problem, Using an online formulation. *Dept. of Mathematics and Computer Science, University of Southern Denmark.* PP-2003-19.

WOLFLER-CALVO, R., DE LUIGI, F., HAASTRUP, P. & MANIEZZO, V. (2004). A distributed geographic information system for the daily car pooling problem. *Computers & Operations Research.* 31 : 13.

ZAMAN, S.U., CHEN, Y.-W. & MIYAGI, H. (2002). GIS Oriented Platform For Solving Real World Logistic Vehicle Routing Problem. *The 2002 International Technical Conference On Circuits/Systems, Computers and Communications.*

ANNEXE. DOCUMENTATION DE LA LIBRAIRIE

Voici la documentation generée par le logiciel NDOC 1.3 à partir de la documentation du code de la librairie.

A.1. Namespace: PolyRoute

A.1.1. PolyRoute Type List

A.1.1.1. Classes

Type	Summary
AlgoArbor	Arborescence de recouvrement de coût minimum.
AlgoBellman	Recherche d'un chemin le plus court.
AlgoCarp	Algorithme de tournées avec capacité.
AlgoCarpRp	Algorithme de tournées avec capacité et points de remplissage
AlgoDijkstra	Algorithme de Dijkstra.
AlgoMaxFlotMinCout	Recherche d'un flot de coût minimal.
AlgoParcoursPC	Recherche d'un parcours eulérien.
AlgoParcoursPR	Recherche d'un parcours passant au moins une fois dans chaque arc obligatoire.
AlgoTSP	Construction de tournées sur les nœuds. (TSP)
ClbAdoNetGeneric	Manipulation de bases de donnés
ClbAttribut	Pour manipuler les Attributs.
ClbChamp	Un champ ou une caractéristique d'un élément d'un graphe.
ClbChamps	Ensemble d'objets de la classe.
ClbMapLiens	Affecter des valeurs aux liens.
ClbMapNL	Classe de base pour l'assignation de valeurs aux clsNoeud ou aux clsLien.
ClbMapNoeuds	Affecter des valeurs aux nœuds
clbStatusBar	Allows an to attempt to free resources and perform other cleanup operations before the is reclaimed by garbage collection.

ClsChemin	Collection d'objets de la classe qui représente un chemin (path) fermé
clsChemins	Collection d'objets de la classe, cet-à-dire une collection de chemins
ClsGraphe	Classe pour la manipulation d'un graphe mathématique.
ClsLien	Classe pour la manipulation des objets (arc, edge) d'un graphe mathématique.
ClsLiens	Collection de Liens
clsList	Liste multi-usage
ClsNL	Classe de base pour des objets élémentaires d'un graphe tel que les Nœuds et les Liens
ClsNoeud	Classe pour la manipulation des objets « nœud » (sommets)
ClsNoeuds	Collection de clsNoeud
clsOptChemin	Manipulation des chemins voir la classe ClsChemin
clsTournee	Manipulation tournées
ClsTroncon	Manipulation Tronçons
clsUtil	Methods multi-usage
colColIntegers	Collection de collections d'entiers.
colColLiens	Collection de collections de liens.
colColNodes	Collection de collections de noeuds.
colGen	Initializes a new instance of the class.
colIntegers	Collection d'entiers.
colLiens	Collection de liens de clsLien
colNodes	Collection de noeuds
colTroncons	Collection d'objets de la classe clsTroncon
modConstants	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

A.1.2. PolyRoute Classes

A.1.2.1. modConstants Class

Summary
public class modConstants

Field Members

Name	Access	Summary
AR_CIRC_LIENS : String	public	

AR_CIRCUITS : String	public	
AR_DISTANCES : String	public	
AR_EDGES : String	public	
AR_NODES : String	public	
AR_NOM_ATTR_CONNECTION : String	public	
AR_NOM_ATTR_DESC_CHEMIN : String	public	
AR_NOM_ATTR_ID_CHEMIN : String	public	
AR_NOM_ATTR_PROVIDER : String	public	
AR_POS_X : String	public	
AR_POS_Y : String	public	
ATR_COST__CUMULEE : String	public	
ATR_DEM__CUMULEE : String	public	
BigDouble : Double	public	
BigInteger : Int32	public	
ci : CultureInfo	public	
MSG_AR : String	public	
rm : ResourceManager	public	
sGm : String	public	

Method Members

Name	Access	Summary

A.1.2.2. clsUtil Class

Constructor Members

Name	Access	Summary
clsUtil()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
afterStr() : String	public	

CAttr() : ClbAttribut	public	
CAttr() : ClbAttribut	public	
Check() : Void	public	

A.1.2.3. colGen Class

Name	Access	Summary
colGen()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	
Item : Object	public	
Item : Object	public	

Method Members

Name	Access	Summary
Add() : Void	public	
Finalize() : Void	protected	
GetEnumerator() : IEnumerator	public	
getFirst() : Object	public	
getLast() : Object	public	
Remove() : Void	public	
Remove() : Void	public	
RemoveAll() : Void	public	

A.1.2.4. ClsTroncon Class

Name	Access	Summary
ClsTroncon()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

aDesservir : Boolean	public	
Arc : ClsLien	public	
attbs : ClbChamps	public	
typeTroncon : AR_TYPETRONCON	public	

Method Members

Name	Access	Summary
Clone() : ClsTroncon	public	
Finalize() : Void	protected	

A.1.2.5. clsTournee Class

Constructor Members

Name	Access	Summary
clsTournee()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
prgStatusBar : clbStatusBar	public	

Method Members

Name	Access	Summary
ParcourirGraphe() : ClsChemin	public	
ParcoursChinois() : ClsChemin	public	
parcoursRural() : ClsChemin	public	
parcoursRural() : ClsChemin	public	
pccBellman() : Int16	public	
pccDijstra() : ClsChemin	public	
ptac() : clsChemins	public	
ptacPR() : clsChemins	public	
ptacPR() : clsChemins	public	
RendreEulerien() : Void	public	

ReverseEulerien() : Void	public	

A.1.2.6. clsOptChemin Class

Constructor Members

Name	Access	Summary
clsOptChemin()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
chemin : ClsChemin	public	
ptrDepot : ClsNoeud	public	

Method Members

Name	Access	Summary
AjouterArcADesservir() : Void	public	
AjouterArcADesservirAlaFin() : Void	public	
ChercherPosDArcADesservir() : Int32	public	
ChercherTronconAOter() : Int32	public	
epargneOterTronconADesservir() : Double	public	
Longueur() : Int32	public	
OterTronconADesservir() : ClsLien	public	
OterTronconsADesservir() : Void	public	
PermuterChemins() : Void	public	
permuterTroncons() : Void	public	
reduireCheminInutile() : Int32	public	
reduireCheminsInutiles() : Void	public	
reduireDistanceParcouru() : Void	public	
remplacerAvecCPP() : Int32	public	
three_Interch() : Void	public	
trouverChaineInutile() : ClsChemin	public	

trouverChainesInutiles() : Collection	public	
trouverPermutation() : ClsChemin	public	
trouverProlongation() : ClsChemin	public	

A.1.2.7. ClsNoeuds Class

Constructor Members

Name	Access	Summary
ClsNoeuds()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
add() : Void	public	Additioner un noeud à la liste de noueds
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
getChamp2Attrib() : ClbAttribut	public	Fonction qui retourne un attribut
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
getNoeud() : ClsNoeud	public	Fonction qui retourne le dernier noeud dans la liste de noeuds
getNoeud() : ClsNoeud	public	Fonction qui retourne la référence du noeud avec l'identificateur du paramètre
isNode() : Boolean	public	Fonction qui retourne si un noeud appartient à la liste de noeuds du graphe
Remove() : Void	public	Effacer un noeud de la liste de noeuds

Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	*(from* PolyRoute.colNodes*)*

A.1.2.8. ClsNoeud Class

Constructor Members

Name	Access	Summary
ClsNoeud()	public	Initializes a new instance of the class.
ClsNoeud()	public	

Property Members

Name	Access	Summary
Attb : Object	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe un champ
Attbs : ClbChamps	public	*(from* PolyRoute.ClsNL*)* La liste complète de champs
Hidden : Boolean	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe la propriété d'être caché
Id : Int32	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe l'identificateur
Owner : ClsGraphe	public	*(from* PolyRoute.ClsNL*)* Le graphe propriétaire

Method Members

Name	Access	Summary
adjEdges() : ICollection	public	Retourne les liens adjacents du noeud
Clone() : ClsNL	public	*(from* PolyRoute.ClsNL*)*
degree() : Int16	public	Le degré du noeud
dpc() : Double	public	Trouve la distance du plus court chemin au
Finalize() : Void	protected	
inDeg() : Int16	public	Le degré des intrants du noeud
inEdges() : ClsLiens	public	Retourne les liens intrants du noeud
isHidden() : Boolean	public	Retourne True si le noeud est caché
Opposite() : ClsNoeud	public	Retourne le noeud du côté opposé

outDeg() : Int16	public	Le degré des extrants du noeud
outEdges() : ClsLiens	public	Retourne les liens extrants du noeud
valAttrib() : Double	public	*(from* PolyRoute.ClsNL*)* Retourne la valeur d'un attribut

A.1.2.9. ClsNL Class

Constructor Members

Name	Access	Summary
ClsNL()	public	Créer une nouvelle instance
ClsNL()	public	Créer une nouvelle instance

Property Members

Name	Access	Summary
Attb : Object	public	Retourne ou fixe un champ
Attbs : ClbChamps	public	La liste complète de champs
Hidden : Boolean	public	Retourne ou fixe la propriété d'être caché
Id : Int32	public	Retourne ou fixe l'identificateur
Owner : ClsGraphe	public	Le graphe propriétaire

Method Members

Name	Access	Summary
Clone() : ClsNL	public	
valAttrib() : Double	public	Retourne la valeur d'un attribut

A.1.2.10. clsList Class

Constructor Members

Name	Access	Summary
clsList()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	

Item : Object	public	

Method Members

Name	Access	Summary
Add() : Void	public	
Contains() : Boolean	public	
getAt() : Object	public	
GetEnumerator() : IEnumerator	public	
getFirst() : Object	public	
getLast() : Object	public	
InsertAt() : Void	public	
InsertAtFirst() : Void	public	
InsertAtLast() : Void	public	
insertSorted() : Int32	public	
insertValueSortedByKey() : Int32	public	
isEmpty() : Boolean	public	
Pop() : Object	public	
PopBack() : Object	public	
Push() : Object	public	
Remove() : Void	public	
RemoveAt() : Void	public	
RemoveFirst() : Void	public	
RemoveLast() : Void	public	
ToString() : String	public	

A.1.2.11. ClsLiens Class

Constructor Members

Name	Access	Summary
ClsLiens()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
add() : Void	public	Additioner un lien
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
getChampToAttrib() : ClbAttribut	public	Fonction qui retourne un attribut
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
getLien() : ClsLien	public	Fonction qui retourne le dernier lien dans la liste de liens
getLien() : ClsLien	public	Fonction qui retourne la reference du lien avec l'identificateur du parametre
isLien() : Boolean	public	Fonction qui retourne si un lien apartient à la liste de liens du graphe
Remove() : Void	public	Effacer un noeud
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	*(from* PolyRoute.colLiens*)*

A.1.2.12. ClsLien Class

Constructor Members

Name	Access	Summary
ClsLien()	public	Initializes a new instance of the class.
ClsLien()	public	

Property Members

Name	Access	Summary
Attb : Object	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe un champ
Attbs : ClbChamps	public	*(from* PolyRoute.ClsNL*)* La liste complète de champs
Hidden : Boolean	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe la propriété d'être caché
Id : Int32	public	*(from* PolyRoute.ClsNL*)* Retourne ou fixe l'identificateur
Owner : ClsGraphe	public	*(from* PolyRoute.ClsNL*)* Le graphe propriétaire
Source : ClsNoeud	public	Le noeud début (la queue) de l'arc
Target : ClsNoeud	public	Le noeud destination (la tête) de l'arc

Method Members

Name	Access	Summary
changeSource() : Void	public	Change le noeud début de l'arc
changeTarget() : Void	public	Change le noeud destination de l'arc
Clone() : ClsNL	public	*(from* PolyRoute.ClsNL*)*
isHidden() : Boolean	public	Retourne True si l'arc est caché
opposite() : ClsNoeud	public	Retourne le noeud du côté opposé
reverse() : ClsLien	public	Cahngement de la direction de l'arc
valAttrib() : Double	public	*(from* PolyRoute.ClsNL*)* Retourne la valeur d'un attribut

A.1.2.13. ClsGraphe Class

Constructor Members

Name	Access	Summary
ClsGraphe()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
attbs : ClbChamps	public	List d'attributs du graphe
ConnectionString : String	public	Retourner le string-connection à la base de données
edgesHidden : ClsLiens	public	Les liens cachés du graphe

isDirected : Boolean	public	Orienter le graph
Liens : ClsLiens	public	Les liens du graphe
Nodes : ClsNoeuds	public	Les noeuds du graphe
nodesHidden : ClsNoeuds	public	Les noeuds cachées du graph
Provider : EnumProviders	public	Retourner le type de provider

Method Members

Name	Access	Summary
chooseNode() : ClsNoeud	public	
composantes() : colColNodes	public	
composantes() : colColNodes	public	
composantesFC() : colColNodes	public	Retourne les composants fortement connexes
composantesFC() : colColNodes	public	Retourne les composants fortement connexes
create() : Void	public	Creer un graphe
create() : Void	public	Creer un graphe
delEdge() : Void	public	
delNode() : Void	public	
EnregistrerAtxt() : Void	public	Enregistrer le graphe dans un fichier texte
ExporterGraphe() : Void	public	Exporter le graph à une base de données Access
Finalize() : Void	protected	
gAcolEdges() : colLiens	public	Copier dans un objet les edges avec l'attribut du parametre
gAedgeMap() : ClbMapLiens	public	Creer un objet à partir d'une collection de liens
getAdoNetGeneric() : ClbAdoNetGeneric	public	Retourner un Ado-Generic de la base de données
getChamp2AttribLien() : ClbAttribut	public	Convertir un string à un objet attribut
getChamp2AttribNoeud() : ClbAttribut	public	Convertir un string à un objet attribut
GetOutEdges() : ClsLiens	public	Fonction qui retourne les liens sortants d'un noeud
graphInverse() : ClsGraphe	public	Calcul du graphe inverse

hideEdge() : Void	public	Cacher un lien
hideNode() : ClsLiens	public	Cacher un noeud
isAcyclic() : Boolean	public	Fonction que retourne Vrai si le graphe est sans cicles
isConnected() : Boolean	public	Fonction que retourne Vrai si le graphe est connexe
isEurelian() : Boolean	public	Fonction que retourne Vrai si le graphe est Eurelian
isStronglyConnected() : Boolean	public	Fonction que retourne Vrai si le graphe est fortement connexe
lireGraphe() : Void	public	Lire un graphe d'une base de données Access ou Excel
lireGrapheSqlServer() : Void	public	Lire un graphe dans une base de données SQL Server
makeDirected() : Void	public	
newEdge() : ClsLien	public	Additionner un lien au graphe
newEdgeAttrib() : ClsLien	public	Additionner un lien et des attributs au graphe
newNode() : ClsNoeud	public	Additionner un nœud au graphe
newNodeAttrib() : ClsNoeud	public	Additionner un nœud et des attributs au graphe
numEdges() : Int32	public	Nombre de liens dans le graphe
numNodes() : Int32	public	Nombre de noeuds dans le graphe
reduireGraphe() : ClsGraphe	public	fonction qui retourne un graphe reduit
restoreEdge() : Void	public	Le lien est mis comme visible dans le graph
restoreGraph() : Void	public	Metre comme visible tous les liens et noeuds cachés dans le graphe
restoreNode() : Void	public	Metre comme visible un noeud
sousGraphePartiel() : ClsGraphe	public	Fonction qui retourne un graphe partial engendré par un sousensemble d'liens
ToString() : String	public	Convertir le graph a une texte, utilise les id de noeuds et liens

A.1.2.14. clsChemins Class

Constructor Members

Name	Access	Summary
clsChemins()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
add() : Void	public	
Add() : Void	public	*(from* PolyRoute.colGen*)*
cheminPlusCourt() : Int32	public	Chemin à moindre coût
Enregistrer() : Void	public	
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
lireChemins() : Void	public	
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
sommeAttribLien() : Double	public	Somme les valeurs du atrribute cible
ToString() : String	public	
ToStringNods() : String	public	

A.1.2.15. ClsChemin Class

Constructor Members

Name	Access	Summary
ClsChemin()	public	Création d'un nouvel objet

Property Members

Name	Access	Summary
attbs : ClbChamps	public	Access aux attributs de l'objet chemin
Count : Int32	public	*(from* PolyRoute.colGen*)*
Description : String	public	Description de cet objet Chemin
Id : Int32	public	Description de cet objet Chemin
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Opt : clsOptChemin	public	

Method Members

Name	Access	Summary
Add() : Void	public	
Add() : Void	public	
Add() : Void	public	*(from* PolyRoute.colGen*)*
ajouterArcPos() : Void	public	Ajouter un Lien dans une position donnée
ajouterTronconPos() : Void	public	Ajouter un troncon dans une position donnée
arcInitial() : ClsLien	public	
arcTerminal() : ClsLien	public	
chercherLien() : Int32	public	Renvoi la position du premier Lien trouvé, 0 s'il n'est pas la
chercherNoeud() : ClsNoeud	public	Chercher la position d'un noeud d'acord aux parametres
chercherPosTroncon() : Int32	public	Chercher la position d'un troncon d'accord aux parametres
Clone() : ClsChemin	public	
connecterLiens() : Void	public	Connecter les arcs du chemin, si ils ne sont pas des arcs adjacents
creerAtrrCum() : Double	public	Creer un nouveau champ avec la somme cumulé d'un autre champ
creerAtrrDemCum() : Double	public	creer le champ de demande cumulé
DeplacerOrigene() : Void	public	Deplacer le noeud initial

desservir() : Void	public	Affecter le propieté aDesservir
Enregistrer() : Void	public	Envoi le chemin à la BD
FermerChemin() : Void	public	Si le chemin est ouvert, c-a-d, le noeud initiel n'est pas le noeud final on conecte le dernier arc au node initial
Finalize() : Void	protected	
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
insererChemin() : Void	public	Inserer un chemin à partir d'une position donée
lireChemin() : Void	public	Lire un Chemin d'une base de données
Node() : ClsNoeud	public	Renvoi le noeud de la position i
Permuter() : Void	public	Permuter deux sous-chemins d'un chemin
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
remplacerChemin() : Void	public	Remplacer un sous-chemin par un Chemin
retNodes() : colNodes	public	renvoi la collection de noeuds
sommeAttribLien() : Double	public	Somme les valeurs du atrribute cible
ToString() : String	public	*(from* PolyRoute.colTroncons*)*
ToStringNods() : String	public	

A.1.2.16. clbStatusBar Class

Constructor Members

Name	Access	Summary
clbStatusBar()	public	

Method Members

Name	Access	Summary

advance() : Void	public	
advance() : Void	public	
current() : Void	public	
DrawItem() : Void	public	
inic() : Void	public	
progress() : Int32	public	
reset() : Void	public	
updatePanel() : Void	public	

A.1.2.17. ClbMapNoeuds Class

Constructor Members

Name	Access	Summary
ClbMapNoeuds()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Item : Object	public	*(from* PolyRoute.ClbMapNL*)*
Keys : ICollection	public	*(from* PolyRoute.ClbMapNL*)*
Values : ICollection	public	*(from* PolyRoute.ClbMapNL*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.ClbMapNL*)* Affecter une valeur au nœud (lien) du paramètre
Clone() : ClbMapNL	public	*(from* PolyRoute.ClbMapNL*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.ClbMapNL*)*
init() : Void	public	Affecter une valeur (la même) a chaque noeud de la liste de noeuds
Remove() : Void	public	*(from* PolyRoute.ClbMapNL*)* Retirer l'affectation de la valeur au nœud (lien) du paramètre
setNewValue() : Void	public	*(from* PolyRoute.ClbMapNL*)* Affecter une valeur au noeud

		(lien) du parametre
wasInit() : Boolean	public	*(from* PolyRoute.ClbMapNL*)* Retourner un boolean pour savoir si le nœud (lien) a déjà une valeur affecté

A.1.2.18. ClbMapNL Class

Constructor Members

Name	Access	Summary
ClbMapNL()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Item : Object	public	
Keys : ICollection	public	
Values : ICollection	public	

Method Members

Name	Access	Summary
Add() : Void	public	Affecter une valeur au nœud (lien) du paramètre
Clone() : ClbMapNL	public	
GetEnumerator() : IEnumerator	public	
Remove() : Void	public	Retirer l'affectation de la valeur au nœud (lien) du paramètre
setNewValue() : Void	public	Affecter une valeur au noeud (lien) du parametre
wasInit() : Boolean	public	Retourner un boolean pour savoir si le nœud (lien) a déjà une valeur affecté

A.1.2.19. ClbMapLiens Class

Constructor Members

Name	Access	Summary
ClbMapLiens()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Item : Object	public	*(from* PolyRoute.ClbMapNL*)*
Keys : ICollection	public	*(from* PolyRoute.ClbMapNL*)*
Values : ICollection	public	*(from* PolyRoute.ClbMapNL*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.ClbMapNL*)* Affecter une valeur au nœud (lien) du paramètre
Clone() : ClbMapNL	public	*(from* PolyRoute.ClbMapNL*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.ClbMapNL*)*
init() : Void	public	Affecter une valeur (la même) à chaque lien de la liste de liens
Remove() : Void	public	*(from* PolyRoute.ClbMapNL*)* Retirer l'affectation de la valeur au nœud (lien) du paramètre
setNewValue() : Void	public	*(from* PolyRoute.ClbMapNL*)* Affecter une valeur au noeud (lien) du parametre
wasInit() : Boolean	public	*(from* PolyRoute.ClbMapNL*)* Retourner un boolean pour savoir si le nœud (lien) a déjà une valeur affecté

A.1.2.20. colIntegers Class

Constructor Members

Name	Access	Summary
colIntegers()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary

Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.21. colNodes Class

Constructor Members

Name	Access	Summary
colNodes()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*

RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.22. colLiens Class

Constructor Members

Name	Access	Summary
colLiens()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.23. colColIntegers Class

Constructor Members

Name	Access	Summary
colColIntegers()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.24. colColNodes Class

Constructor Members

Name	Access	Summary
colColNodes()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary

Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.25. colColLiens Class

Constructor Members

Name	Access	Summary
colColLiens()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.colGen*)*
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*

RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.26. colTroncons Class

Constructor Members

Name	Access	Summary
colTroncons()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Count : Int32	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*
Item : Object	public	*(from* PolyRoute.colGen*)*

Method Members

Name	Access	Summary
Add() : Void	public	*(from* PolyRoute.colGen*)*
Clone() : colTroncons	public	
Finalize() : Void	protected	*(from* PolyRoute.colGen*)*
GetEnumerator() : IEnumerator	public	*(from* PolyRoute.colGen*)*
getFirst() : Object	public	*(from* PolyRoute.colGen*)*
getLast() : Object	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
Remove() : Void	public	*(from* PolyRoute.colGen*)*
RemoveAll() : Void	public	*(from* PolyRoute.colGen*)*
ToString() : String	public	

A.1.2.27. ClbChamps Class

Constructor Members

Name	Access	Summary

ClbChamps()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Item : Object	public	
Keys : ICollection	public	
Values : ICollection	public	

Method Members

Name	Access	Summary
Add() : Void	public	
Clone() : ClbChamps	public	
getChamp2Attrib() : ClbAttribut	public	
isChamp() : Boolean	public	
majAttrib() : Void	public	
Remove() : Void	public	
RemoveAll() : Void	public	

A.1.2.28. ClbChamp Class

Constructor Members

Name	Access	Summary
ClbChamp()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Nom : String	public	Le nombre du champ
Valeur : Object	public	Retourner ou assigner la valeur qui tendra l'object

Method Members

Name	Access	Summary
Clone() : ClbChamp	public	

Finalize() : Void	protected	

A.1.2.29. ClbAttribut Class

Constructor Members

Name	Access	Summary
ClbAttribut()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
Attrib : Object	public	Retourner ou assigner l'attribut

Method Members

Name	Access	Summary
Clone() : ClbAttribut	public	
setNewValue() : Void	public	Additioner un attribut
valeur() : Double	public	Retourner la valeur de l'attribut
verifierExistence() : Void	public	verifier l'existence si le champs n'existe pas, une exception est generée

A.1.2.30. ClbAdoNetGeneric Class

Constructor Members

Name	Access	Summary
ClbAdoNetGeneric()	public	Crear un composant generique
ClbAdoNetGeneric()	public	Crear un composant Access ou Excell

Property Members

Name	Access	Summary
ConnectionString : String	public	Retourner le string-connection à la base de données
Provider : EnumProviders	public	Retourner le type de provider

Method Members

Name	Access	Summary

fillDataTable() : DataTable	public	Remplir une DataTable
fillSchemaDataSet() : DataSet	public	Remplir un SchemaDataSet
Finalize() : Void	protected	
GetCommand() : IDbCommand	public	Retourner le Command
GetCommandBuilder() : Object	public	Retourner le xxxCommandBuilder
GetConnection() : IDbConnection	public	Retourner la connection
GetDataAdapter() : IDbDataAdapter	public	Retourner le DataAdapter
GetTables() : Boolean	public	Retourner les nom de tablas de un base de données
GetTables() : DataTable	public	
openConnection() : IDbConnection	public	Ouvrir une connection à une base de données
ReadMyData() : OleDbDataReader	public	
updateDataTable() : Void	public	Metre à jour une DataTable

A.1.2.31. AlgoTSP Class

Constructor Members

Name	Access	Summary
AlgoTSP()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
check() : Int32	public	
classerLongueurs() : Void	public	
defParametres() : Void	public	
deltaCosto4opt() : Double	public	
Finalize() : Void	protected	
perm4() : Void	public	
reset_Renamed() : Void	public	
retLstNodes() : colNodes	public	
retParcours() : ClsChemin	public	

run4opt() : Int32	public	
runGreedy() : Int32	public	
runSpaceFilling() : Int32	public	
updateRuta4opt() : Void	public	

A.1.2.32. AlgoParcoursPR Class

Constructor Members

Name	Access	Summary
AlgoParcoursPR()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
prgStatusBar : clbStatusBar	public	
utiliserVraiDistance : Boolean	public	

Method Members

Name	Access	Summary
check() : AR_ALGORESULTS	public	
defParametres() : Void	public	
Finalize() : Void	protected	
retParcours() : ClsChemin	public	
run() : AR_ALGORESULTS	public	

A.1.2.33. AlgoParcoursPC Class

Constructor Members

Name	Access	Summary
AlgoParcoursPC()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
check() : Int32	public	

defParametres() : Void	public	
estEurelian() : Boolean	public	
Finalize() : Void	protected	
RendreEurelian() : Void	public	
retParcours() : ClsChemin	public	
ReverseEurelian() : Void	public	
run() : Int32	public	

A.1.2.34. AlgoMaxFlotMinCout Class

Remarks
Algorithme de Busacker

Constructor Members

Name	Access	Summary
AlgoMaxFlotMinCout()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
prgStatusBar : clbStatusBar	public	

Method Members

Name	Access	Summary
check() : Int32	public	
CoutTotal() : Double	public	
defParametres() : Void	public	
Finalize() : Void	protected	
FluxTotal() : Double	public	
FluxTotalEdge() : Double	public	
FluxTotalEdges() : ClbMapLiens	public	
reset() : Void	public	
run() : Int32	public	
runBellman() : Int32	public	

runFifo() : AR_ALGORESULTS	public	

A.1.2.35. AlgoDijkstra Class

Remarks
Algorithme de Dijkstra

Constructor Members

Name	Access	Summary
AlgoDijkstra()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
distance : Single	public	
predEdge : ClsLien	public	
predNode : ClsNoeud	public	
reached : Boolean	public	
shortDistance : Single	public	

Method Members

Name	Access	Summary
check() : AR_ALGORESULTS	public	
defParametres() : Void	public	
Finalize() : Void	protected	
reRun() : Int16	public	
run() : Int16	public	
shortPath() : ClsChemin	public	

A.1.2.36. AlgoCarpRp Class

Remarks
Algorithme de tournées avec capacité et points de remplissage

Constructor Members

Name	Access	Summary

AlgoCarpRp()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
check() : AR_ALGORESULTS	public	
ConstruirGrapheTransforme() : ClsGraphe	public	
defParametres() : Void	public	Parametres de l'algorithme de tournés de vehicules [1,1,-]
dernierEnsemble() : Boolean	public	
retChemins() : clsChemins	public	
retPointsRemplissage() : colNodes	public	
runAmaPaper2() : Int32	public	
runCut() : Int32	public	
runCutTSP() : Int32	public	
trouverPointsEnsembleNi() : colIntegers	public	
trouverSuivanteEnsembleNi() : colIntegers	public	
truverEnsemblesNodes_Netoile() : colColIntegers	public	

A.1.2.37. AlgoCarp Class

Remarks
Algorithme du CARP

Constructor Members

Name	Access	Summary
AlgoCarp()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
check() : Int16	public	
ConstruireGraphe() : ClsGraphe	public	
contientDepot() : Boolean	public	
defParametres() : Void	public	

retChemins() : clsChemins	public	
retTournée() : ClsChemin	public	
runCut() : Int16	public	
runCutMonitoring() : Int16	public	
runPearn() : Int16	public	
runUlusoy() : Int16	public	
trouverPGindice() : Int16	public	

A.1.2.38. AlgoBellman Class

Remarks
Algorithme de Bellman

Constructor Members

Name	Access	Summary
AlgoBellman()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
source : ClsNoeud	public	
storePreds : Boolean	public	
weights : ClbAttribut	public	

Method Members

Name	Access	Summary
check() : Int16	public	
defParametres() : Void	public	
distance2Node() : Single	public	
distances() : ClbMapNoeuds	public	
Finalize() : Void	protected	
negativeCycle() : Boolean	public	
predecessorEdge() : ClsLien	public	
predecessorNode() : ClsNoeud	public	

predecessors() : ClbMapNoeuds	public	
reached() : Boolean	public	
reset() : Void	public	
run() : Int16	public	
runFifo() : Int16	public	

A.1.2.39. AlgoArbor Class

Remarks
Algorithme de Prins

Constructor Members

Name	Access	Summary
AlgoArbor()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
source : ClsNoeud	public	
weights : ClbAttribut	public	

Method Members

Name	Access	Summary
check() : Int32	public	
defParametres() : Void	public	
Finalize() : Void	protected	
minArbor() : ClsLiens	public	
minCoutArbor() : Int32	public	
nbRecouverts() : Int32	public	
nearest() : ClsLien	public	
reset() : Void	public	
run() : Int32	public	